Trabajo de Grado

Integrando telefonía IP con una aplicación de gestión de tiempos

Investigación Aplicada y Desarrollo Tecnológico
Director: Díaz, Francisco Javier
Codirector: Queiruga, Claudia Alejandra
Alumnos: Butierrez, Sebastián O. (1641/6)
Ramos Giacosa, Luis F. (1151/2)

Facultad de Informática
Universidad Nacional de La Plata
2007

Índice

Introducción

La Telefonía IP es una tecnología de transmisión de servicios de comunicación que requiere una única red IP, pública o privada, en lugar de redes de voz, video y datos por separado. La idea base de la Telefonía IP fue la eliminación de los costos en las llamadas telefónicas de larga distancia internas o externas, haciendo uso de la digitalización de la voz y enviándola en forma de paquetes IP a través de Internet o de una intranet.

El auge de la Telefonía IP tuvo sus inicios en los alrededores de 1995 cuando se introdujo el primer producto de software de teléfono, diseñado para correr en una PC con placa de sonido, parlantes, micrófono, módem y capaz de comprimir la señal de voz y traducirla en paquetes IP para después trasmitirla a través de Internet [42]. De esta manera, se pudo establecer una "comunicación telefónica" de PC a PC a través de Internet, sin necesidad de efectuar la llamada de larga distancia que hubiese requerido una comunicación en la red de telefonía tradicional.

Desde entonces, el desarrollo de esta tecnología fue avanzando rápidamente, pasando por productos de software más complejos, por gateways capaces de actuar como interfaz entre Internet y la red de telefonía estándar y por el advenimiento de dispositivos de un rango completo de inteligencia de comunicación como son los Teléfonos IP. Estos aparatos brindan la posibilidad de conectarse a una red IP, proveyendo la misma funcionalidad que un teléfono convencional, como así también, un conjunto de características más complejas como, por ejemplo, el acceso a Internet.

En la actualidad se sigue avanzando en la definición de estándares y en la investigación de cambios en las líneas de comunicación. Estas continuas búsquedas de mejoras en la calidad del servicio han llevado a la posibilidad de interacción con aplicaciones de propósito general que puedan requerir ser accedidas en cualquier momento o lugar. Un estilo de aplicación que requiere acceso universal es una agenda o gestor de tiempos.

Los usuarios de este tipo de herramienta se encuentran generalmente limitados en el acceso a la información dado que sólo pueden hacerlo desde la red corporativa de su organización.

Mediante la utilización de una aplicación que **integre** la telefonía IP y la herramienta de gestión de tiempos, se puede facilitar el acceso a la información desde cualquier lugar a través de una red IP como Internet.

De esta manera los usuarios del gestor de tiempos tendrán la posibilidad de manejar los datos haciendo uso de la potencialidad brindada por los teléfonos IP, dado que estos dispositivos se pueden conectar a cualquier red IP, proporcionan la navegabilidad necesaria para manejar la información y utilizan estándares de comunicación.

El objetivo de este trabajo fue plasmar esa necesidad de **integración** en el desarrollo de una aplicación Web para teléfonos IP de Cisco Systems basada en Servicios Web que actúe como nexo entre estos dispositivos y una herramienta de Gestión de tiempos, en particular Lotus Organizer.

La información que se buscó manejar comprende los datos referidos a citas, recordatorios, direcciones, contactos, etc. Por lo cual, con la tecnología provista por los teléfonos IP se logró poder consultar, crear y modificar la información manejada en el Lotus Organizer.

Así, mediante el desarrollo de una **solución integradora**, se pretendió facilitar al usuario el acceso a la información del Lotus Organizer que habitualmente usa en su red corporativa, haciendo uso de dispositivos innovadores y en creciente expansión como son los teléfonos IP. Asimismo, se buscó diseñar una arquitectura basada en estándares que sirva como modelo a otras aplicaciones o servicios que puedan ser utilizados desde teléfonos IP para acceder a información corporativa o global.

Para poder lograr la interacción entre los teléfonos IP de Cisco Systems y la herramienta Lotus Organizer de IBM, se desarrollaron un Servicio Web y una aplicación Web sobre la plataforma .NET de Microsoft, utilizando el lenguaje de programación C#, generando un componente COM+ y usando una API de Lotus Organizer. Esta diversidad de fabricantes y tecnologías originó la aparición de un conjunto de interfaces entre estos elementos y la necesidad de utilización de estándares como XML, SOAP y HTTP. Como consecuencia de todo ello, se arribó a una **solución innovadora**, ya que se siguieron tendencias actuales y se utilizaron tecnologías en auge y en constante crecimiento como lo son la telefonía IP, el uso de los estándares mencionados, la plataforma .NET y los Servicios Web, todos ellos elementos fundamentales de muchos desarrollos tecnológicos y plataformas de la actualidad y futuras.

1. Telefonía IP

La telefonía IP es una tecnología que permite la convergencia de servicios de voz, datos y vídeo en una sola red. El principal objetivo apunta al área de voz (VoIP – Voz sobre el Protocolo de Internet) y su potencial integración con las otras dos tecnologías.

La telefonía IP es el conjunto de tecnologías que habilitan comunicaciones de voz, datos y vídeo a través de LAN y WAN basadas en IP e Internet. Utiliza los estándares abiertos IETF e ITU para trasmitir multimedia a través de cualquier red que utilice el Protocolo de Internet (IP) y permite la integración sin problemas de comunicaciones de datos, audio y vídeo. El protocolo IP (Internet Protocol) es el método usado para enviar la información desde una computadora a otra; cada mensaje de datos es particionado en pequeñas porciones llamados paquetes [4] [22].

Las comunicaciones telefónicas se efectúan a través de redes TCP/IP y, a diferencia de la Red Pública de Telefonía PSTN (Public Switched Telephone Networks), que se compone de señales analógicas y digitales a través de una red con conmutación de circuitos (circuit-switched), la telefonía IP utiliza conmutación de paquetes (packet-switched) para intercambiar voz, video, fax y otros tipos de información [22][23]. Los teléfonos, videoteléfonos, localizadores, dispositivos de fax dedicados, controladores PBX (centralita) y otros dispositivos de hardware conectados a la PSTN o a una PBX controlan los servicios de telefonía tradicionales. Con la integración de equipos y teléfonos, los servidores de red son compatibles con sistemas PBX y los equipos pueden funcionar como teléfonos, dispositivos de fax y contestadores automáticos [4].

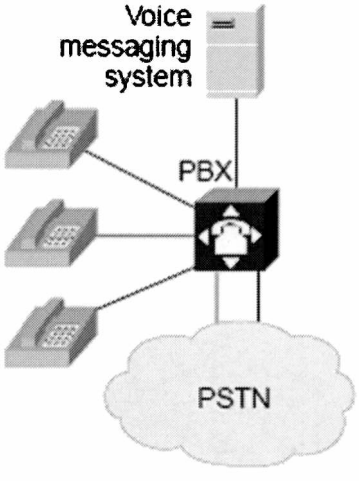


Figura 1. Red convencional de voz

Con la telefonía IP, toda la información que se va a transmitir a través de la red se divide en paquetes de datos. Cada paquete tiene un encabezado que contiene el origen y el destino, un número de secuencia, un bloque de datos y un código de comprobación de errores. Los enrutadores y los servidores dirigen estos paquetes a

través de la red hasta que llegan a su destino. Cuando llegan los paquetes, se utiliza el número de secuencia para volver a juntarlos en su orden original. A diferencia de la telefonía PSTN, que dedica un circuito a una llamada de teléfono, los paquetes de datos comparten un circuito con otras transmisiones [4].

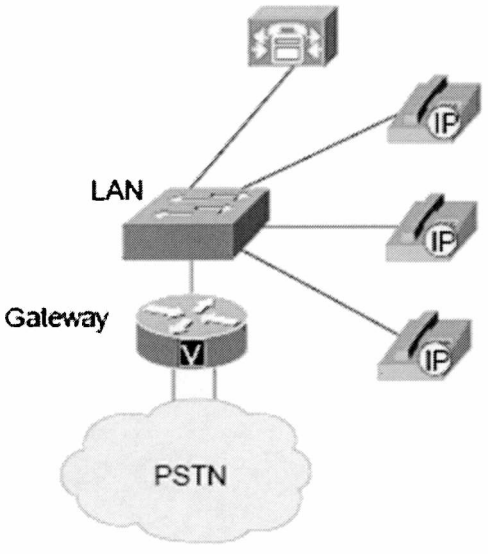


Figura 2. Red IP

PSTN utiliza un ancho de banda fijo para cada conversación y por eso la calidad está garantizada. Cuando el llamador efectúa una llamada, toma el teléfono, escucha el tono, marca el número, la oficina central establece la conexión y, entonces, el receptor de la llamada junto con el llamador hablan uno con el otro. Cuando el llamador hace una llamada mediante Telefonía IP, levanta el teléfono y escucha el tono de la Centralita Telefónica (PBX - Private Branch Exchange), después marca el número que es enviado al gateway de Telefonía IP más cercano ubicado entre la PBX y la red TCP/IP. El teléfono IP encuentra una ruta a través de Internet, llega al número marcado y, finalmente, se establece la llamada. El gateway de Telefonía IP modula la voz en paquetes IP y los envía a través de la red TCP/IP tal como envía los típicos paquetes de datos. Una vez que el gateway remoto de Telefonía IP recibe los paquetes de voz codificados, los reensambla en señales analógicas para el receptor mediante la PBX [23].

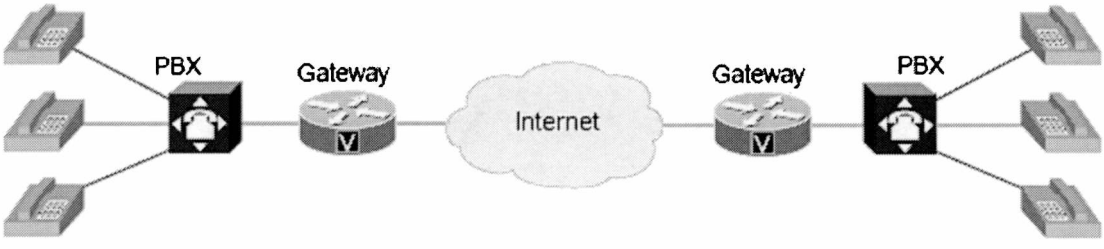


Figura 3. Red IP de Telefonía usando teléfonos convencionales

Los sistemas de telefonía basados en LAN son útiles tanto para organizaciones pequeñas como para grandes organizaciones. La ubicación física de los dispositivos de telefonía no es importante, por lo que el sistema de telefonía basado en LAN funciona correctamente en un entorno distribuido, como las compañías que tienen sitios remotos conectados a su oficina principal mediante una red de área extensa (WAN)

El resultado de todo esto es una red IP que provee una plataforma de transporte común para voz, video y tráfico integrado de datos. Esto apunta a compañías que buscan beneficios de una solución eficiente, rentable y escalable para así unirse con los requerimientos de negocios de hoy en día.

1.1. Calidad del servicio para telefonía IP

Las redes con conmutación de paquetes como Internet no dedican una conexión entre partes, por lo que no pueden garantizar la calidad del servicio (QoS – Quality of Service), al contrario que PSTN. Desde su creación se han utilizado para programas, como correo electrónico y transferencia de archivos, donde una calidad de servicio variable era aceptable.

Sin embargo, la calidad de servicio de telefonía IP está mejorando constantemente, incluso en Internet. La falta de confiabilidad de Internet no se debe al protocolo TCP/IP, sino a que hay muchas redes conectadas que no tienen una autoridad única para asignar recursos. En períodos de uso prolongados, los usuarios envían paquetes adicionales a redes que ya están saturadas.

Para asegurar la calidad de voz sobre enlaces con ancho de banda limitado, la red usa la característica de QoS, la cual clasifica el tráfico de voz y datos en diferentes categorías y satisface la entrega de paquetes de voz en tiempo real. Básicamente, es una estrategia de colas que permite dar prioridad a tráfico de voz y video. El principal objetivo de QoS es ayudar a reducir o eliminar el retardo de los paquetes de voz que viajan a través de la red.

Ahora las organizaciones pueden designar redes privadas para proporcionar el ancho de banda apropiado y rutas superfluas entre puntos de conmutación para conseguir una telefonía IP de alta calidad. Los nuevos productos de software y hardware han mejorado la calidad de servicio, y la transmisión de voz mediante IP a través de una WAN puede ser tan buena como la de las llamadas PSTN estándar, incluso si la red está congestionada.

1.2. Ventajas

Los beneficios de la telefonía IP pueden ser divididos de la siguiente manera:

1- Reducción de costos [12] [22]:

- la reducción de los costos en llamadas de larga distancia es siempre importante y una buena razón para introducir VoIP.

- La telefonía IP reduce el costo de los servicios existentes, como voz y difusión de vídeo, y amplía los medios de comunicación al incluir audio conferencia y videoconferencia.

- Es menos costosa la mudanza de teléfonos dentro de la compañía.

- Cableado y hardware único para soportar tanto voz como datos.

2- Simplificación: una infraestructura integrada que soporta todas las formas de comunicación, permite una mayor estandarización y reduce la cantidad de equipamiento. Esta infraestructura combinada puede soportar optimización dinámica del ancho de banda y un diseño tolerante a fallas. Las diferencias entre los patrones de tráfico de voz y datos ofrecen futuras oportunidades para incrementos significativos de eficiencia [12] [22].

3- Consolidación: debido a que las personas son los elementos más costosos en una red, entonces cualquier oportunidad de combinar operaciones, eliminar puntos de falla y consolidar planes de cuentas va a ser beneficioso.

4- Aplicaciones avanzadas: aunque la telefonía básica y el fax son las aplicaciones iniciales para VoIP, los beneficios a largo plazo se espera que sean a partir de aplicaciones multimedia y multi-servicio. Por ej., una solución de comercio por Internet puede combinar acceso Web a la información con un botón para llamadas que permita acceso inmediato a un agente de un centro de llamadas desde la PC. Combinando características de voz y datos dentro de nuevas aplicaciones va a proveer los mayores beneficios a largo plazo.

5- Más servicios: A medida que la telefonía IP crezca en popularidad va a haber un incremento en el número de aplicaciones y características provistas

- Arquitecturas abiertas y dispositivos no propietarios, sumado a la utilización de estándares permitirá un incremento en la creación de aplicaciones.
- La cantidad de dispositivos que van a proveer servicios de voz va a ir aumentando dramáticamente.
- Va a ser más fácil integrar características de comunicación como páginas Web, teléfonos, pagers, voice-mail, e-mail, etc.

Por ejemplo, un teléfono IP podría actuar como cliente de una aplicación Web. En este caso, para la comunicación existirían los siguientes pasos [9]:

a) El teléfono IP solicita una página Web.
b) El servidor Web toma el requerimiento y dinámicamente genera una página para retornar al teléfono.
c) La página Web resultante es enviada al teléfono.
d) El teléfono IP parsea el archivo recibido, lo interpreta y determina cómo va a ser mostrado en su display.

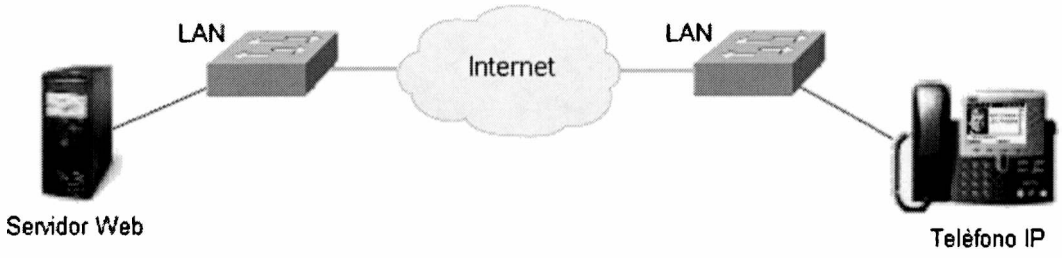


Figura 4. Teléfono IP actuando como cliente de una aplicación Web

En este último punto se hace hincapié durante el desarrollo de este proyecto, dado que hemos logrado la integración entre una aplicación de Gestión de tiempos y la telefonía IP, mediante la utilización de una aplicación Web, un servicio Web y estándares como XML y SOAP.

1.3. Tendencias de la telefonía IP

Durante el transcurso de la década pasada, hubo cambios en la manera en que las personas y las organizaciones se han estado comunicando. Muchos de esos cambios provienen del crecimiento de Internet y de las aplicaciones basadas en IP. En el inicio de estos avances tecnológicos, las soluciones de tráfico de voz y datos son las aplicaciones que más han tomado ventaja del protocolo IP. Debido a que las empresas reciben nuevas y crecientes demandas de los clientes en busca de mayor accesibilidad a la información, la telefonía IP es una importante tecnología en crecimiento y que soporta las operaciones de negocio. Los IT y gerentes de negocios encuentra de utilidad el conocimiento acerca de cómo otros están desarrollando y manejando los sistemas de tecnologías de voz y datos, para así rever las tendencias corrientes en los desarrollos de Telefonía IP y aprender qué facetas y características de la Telefonía IP encuentran esenciales sus pares para proveer a sus compañías con soluciones integradas de voz y datos sobre IP y, así, no perder competitividad frente a los demás [38].

Para entender mejorar estas tendencias se pueden explorar algunos aspectos claves de cómo los negocios están implementando la telefonía IP.

Uno de los puntos a analizar es lo que concierne a las prácticas actuales de implementación y desarrollo de las tecnologías de voz y datos. Este aspecto está limitado por el conocimiento acerca de las opciones y capacidades de los sistemas. Mientras que muchos profesionales IT están implementando telefonía IP en las distintas sucursales de las empresas, la mayoría se encuentran usando la tecnología PBX tradicional provista por un único proveedor. Algunos de los cuales están imposibilitados de implementar una solución de telefonía IP, mientras que otros necesitan de cambios moderados o sustanciales para usar las capacidades de la telefonía IP [38].

Otro aspecto a analizar, son las tendencias existentes en la implementación y desarrollo de tecnologías de convergencia entre voz y datos. Aquí los datos revelan que

la tendencia es implementar telefonía IP en el corto plazo en aquellas organizaciones que aún no lo han hecho y tiene la capacidad de hacerlo [38].

También se deben tener en cuenta aspectos críticos y características de la tecnología de voz y datos. Dentro de este punto, se debe analizar consideraciones críticas y características esenciales de los proveedores de tecnología de voz y datos y de telefonía IP. Las dos consideraciones más críticas para las comunicaciones de voz son la confiabilidad y la calidad. Los atributos más importantes de un proveedor de telefonía IP son que posea tecnologías de voz confiables y de alta calidad, flexibilidad en configuraciones y servicios, y que tenga tecnologías de red confiables y de alta calidad [38].

No hay duda que la telefonía IP está haciendo incursiones considerables dentro de las infraestructuras de comunicación de las empresas. Aunque muchos encargados de negocios y tecnologías aún no conocen la variedad existente de soluciones de convergencia de voz y datos que están disponibles para resolver sus necesidades [38].

1.4. El teléfono IP

Un teléfono IP es un teléfono que provee comunicación de voz a través de la misma red de datos que utilizan las computadoras. Este aparato funciona como un teléfono analógico tradicional permitiendo efectuar y recibir llamadas. Además soporta otras características que cualquiera puede esperar de un teléfono, tales como discado rápido, rellamado, transferencia de llamadas, conferencias y acceso a correo de voz [8].

Como forma parte de una red IP, el teléfono puede proveer no sólo servicios telefónicos, sino también servicios de datos, incluyendo información al minuto y servicios provistos en la red. Es decir, un teléfono IP provee un conjunto completo de características de comunicación que no se esperan en un teléfono tradicional.

Un teléfono IP típico posee una pantalla LCD que muestra características como fecha, hora, número de teléfono, caller ID, estado de línea y permite visualizar servicios provistos para el mismo. Además posee botones que permiten navegar las pantallas provistas por los servicios, acceder a datos de mensajes de voz, de llamadas, de contactos y mostrar información de ayuda [8].

El teléfono provee la posibilidad de acceder a servicios de información como estado del clima, o bien, algún servicio particular brindado en la compañía y navegar en la información utilizando los botones y la pantalla LCD. Estos servicios pueden ser brindado a través de una aplicación Web, siendo el teléfono capaz de conectarse por el protocolo HTTP a la misma [9].

1.5. Servicios para teléfonos IP

En el desarrollo de servicios para teléfonos IP se deben tener en cuenta un conjunto de características particulares que tienen los teléfonos. Para describirlas

tomamos como referencia los teléfonos IP de Cisco Systems utilizados para este proyecto.

Los modernos teléfonos IP de Cisco pueden ser utilizados para desarrollar servicios con los cuales los usuarios puedan interactuar vía el teclado y la pantalla del mismo. Estos servicios son utilizados usando el protocolo HTTP desde un servidor Web estándar.

Algunos servicios típicos que podrían ser provistos para un teléfono IP son [9]:

- Clima
- Información de Stock
- Información de contactos
- Noticias de la compañía
- Agenda

La forma en que un servicio es configurado determina si se muestra un menú gráfico o de texto. Un usuario del teléfono puede navegar un menú de texto o gráfico usando las teclas numéricas o de navegación.

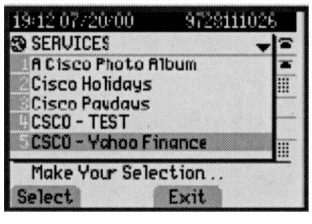


Figura 5. Ejemplo de un menú de texto en un teléfono IP de Cisco

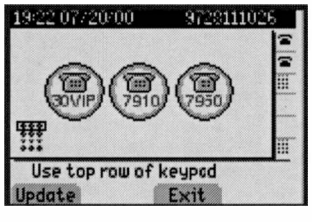


Figura 6. Ejemplo de un menú gráfico en un teléfono IP de Cisco

Una vez que es elegida una opción de menú, el teléfono IP actúa como cliente HTTP para poder cargar una URL específica. El resultado de esa URL puede ser texto plano o un XML específico del tipo de dato CiscoIPPhone XML (conjunto de XMLs interpretables por el teléfono). El teléfono carga el objeto y luego interactúa con el usuario de la manera apropiada para ese tipo de dato [9].

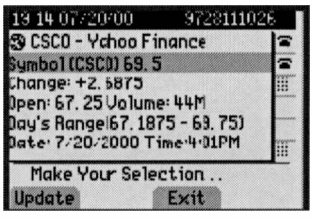


Figura 7. Visualización de un texto plano en la pantalla

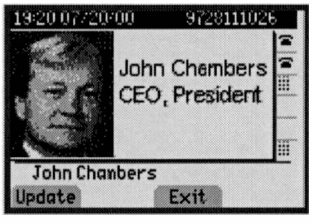


Figura 8. Visualización de una imagen gráfica

La actividad del servicio del teléfono IP de Cisco está limitada al panel específico de la pantalla. Un servicio no puede modificar la línea más alta del display, la cual contiene por ej., la fecha y la hora. Tampoco puede modificar la línea más baja del display, la cual contiene la definición de ciertas teclas.

Un desarrollador de un servicio para estos teléfonos, debe recordar que el teléfono no es un Web browser por naturaleza y no es capaz de parsear HTML. A pesar de que el contenido es enviado al teléfono por un servidor Web, este contenido no es HTML. Todo el contenido es enviado como texto plano o empaquetado dentro de XMLs propietarios [9].

Una cosa importante a tener en cuenta sobre los servicios, es que los teléfonos no tienen ningún tipo de estado cuando cargan una página XML. Los teléfonos IP pueden usar HTTP para cargar el contenido de la página de diferentes maneras, dependiendo de qué tipo de dato (tipo de XML) contiene la página.

El servidor Web debe enviar las páginas XML con tipo MIME del formato text/xml. El mecanismo exacto requerido varía de acuerdo al tipo de Web server que se esté utilizando y el mecanismo del lado del servidor que se utilice para crear las páginas, por ejemplo, archivos estáticos, JavaScript, CGI, ASP, etc [9].

2. COM-DCOM

COM es un modelo de programación basado en objetos diseñado para promover interoperabilidad del software; permite que dos o más aplicaciones o componentes cooperen fácilmente el uno con el otro, incluso si fueron escritos por distintos vendedores, en diferentes momentos, en diversos lenguajes de programación, o si están funcionando en distintas computadoras que corren diferentes versiones de la familia de sistemas operativos de Microsoft Windows [4].

COM (Component Object Model) permite que un objeto exponga su funcionalidad a otros componentes y aloje aplicaciones. Define cómo el objeto se expone a sí mismo y cómo funciona dicha exposición en procesos y en redes. Además, COM define el ciclo de vida del objeto.

Los siguientes conceptos son fundamentales para COM:

- **Interfaces**: mecanismo mediante el cual un objeto expone su funcionalidad.
- **IUnknown**: interfaz básica sobre la que se basan las demás. Implementa el recuento de referencia y los mecanismos de consulta de interfaz que se ejecutan mediante COM.
- **Recuento de referencia**: técnica mediante la cual un objeto (o, más estrictamente, una interfaz) decide cuándo ya no se está utilizando y, por tanto, es libre de eliminarse a sí mismo.
- **QueryInterface**: método utilizado para consultar un objeto para una interfaz determinada.
- **Cálculo de referencias**: mecanismo que permite que los objetos se utilicen más allá de los límites de subproceso, proceso y red, lo que permite una independencia de la ubicación.
- **Agregación**: modo en que un objeto puede hacer uso de otro.

2.1. De sistemas de dos capas a sistemas de n-capas

Una de las principales razones para utilizar COM consiste en desplazar los sistemas de información de una compañía de una arquitectura de dos capas a una arquitectura de n-capas. Esta transición requiere el diseño y la creación de una capa de nivel medio de objetos de negocio. Generalmente los objetos de negocio se sitúan entre las aplicaciones cliente y los servidores de bases de datos. COM sirve como plataforma para estos tipos de sistemas.

Los sistemas de dos capas se han desarrollado ampliamente en la industria, por lo que los problemas asociados a los mismos son bastante conocidos. Examinemos los defectos clave de la arquitectura de dos capas tradicional como la que se muestra en la siguiente figura.

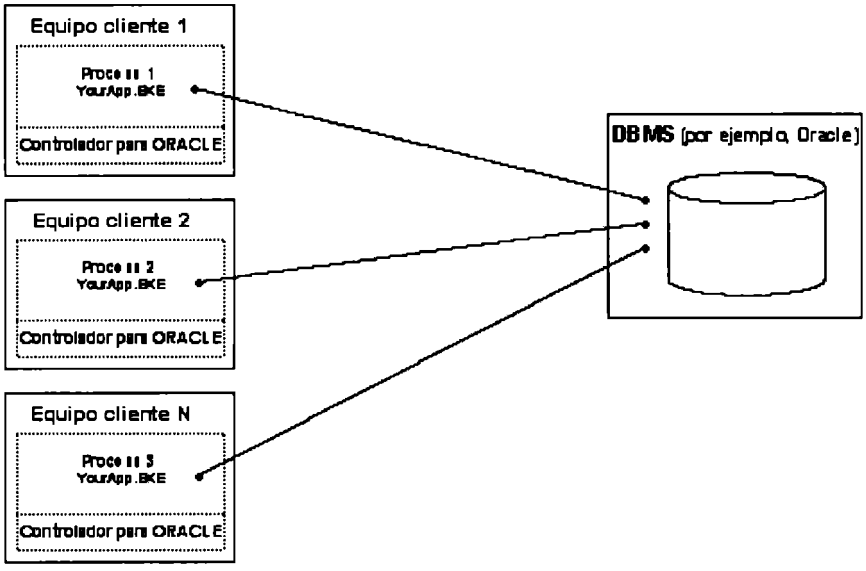


Figura 9. Las aplicaciones de dos capas típicas requieren una conexión independiente a un servidor de bases de datos para cada usuario y el equipo del usuario debe disponer del controlador adecuado para un sistema específico de administración de bases de datos (DBMS).

- El código de la interfaz de usuario se mezcla con el código que implementa la lógica de negocio y de acceso a datos, con lo cual se dificulta volver a usarlo en diferentes aplicaciones cliente. Además, las modificaciones que se realizan en la lógica de negocio o en la base de datos implican a menudo volver a construir y distribuir las aplicaciones cliente [15].
- Cada aplicación cliente, junto con el código que implementa la lógica de negocio y de acceso a datos, se ejecuta en un proceso separado por lo que no se pueden compartir recursos específicos de proceso como subprocesos (threads) y memoria. Asimismo, no se puede compartir una conexión a la base de datos en un sistema de dos capas debido a que cada usuario requiere una conexión separada con el servidor de bases de datos. Ocurre lo mismo con las aplicaciones cliente que se conectan con un gran sistema (mainframe). Esta incapacidad de compartir recursos eficazmente entre un grupo de usuarios limita el rendimiento y la escalabilidad generales de un sistema [15].
- Cada equipo cliente necesita uno o más controladores propietarios para que pueda comunicarse con un servidor de bases de datos o una aplicación de mainframe, lo cual produce un aumento en el costo de mantenimiento y configuración de los equipos de escritorio. El administrador del sistema debe instalar y mantener un conjunto de controladores en cada equipo para proporcionar el acceso a las bases de datos (ODBC). Si se necesita cambiar el motor de base de datos, por ejemplo, cuando se migra de Sybase a Oracle, el administrador debe visitar cada uno de los equipos cliente. Esta operación resulta costosa.
- Las aplicaciones cliente encuentran dificultades para tener acceso a los datos desde varios orígenes de datos. En un entorno empresarial es normal que los datos importantes se extiendan a través de varios sistemas tal y como se muestra en la Figura 10. La situación se complica cuando una aplicación necesita ejecutar distintas transacciones de negocio utilizando datos que están

distribuidos en varios equipos. La situación se vuelve extremadamente difícil cuando los datos también se almacenan en distintos formatos.

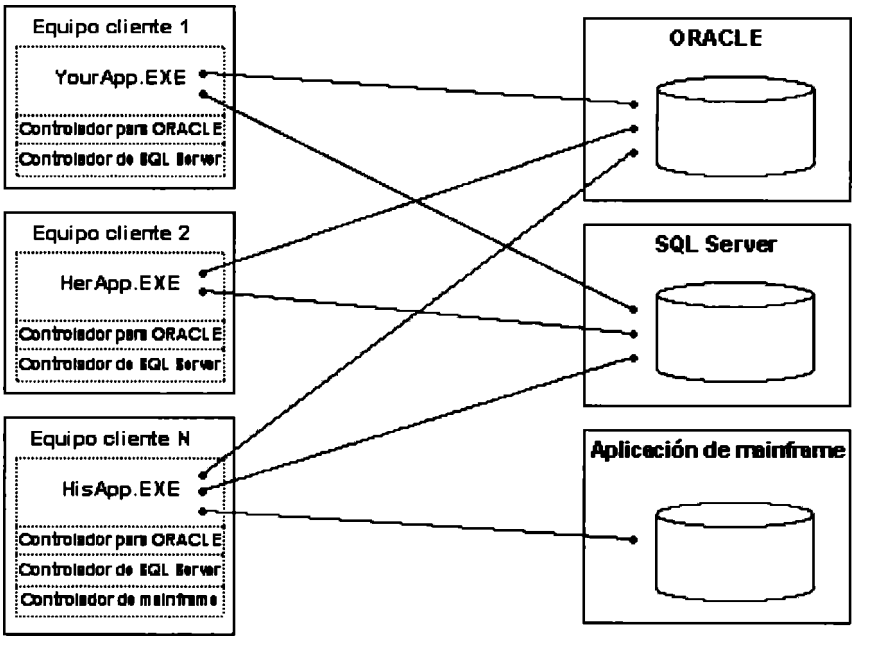


Figura 10. Una estrategia de dos capas no funciona correctamente cuando una compañía dispone de varias aplicaciones cliente o varios servidores de bases de datos.

- Con frecuencia, es necesario que las aplicaciones cliente se ejecuten en la misma red de área local (LAN) que el servidor de bases de datos o la aplicación de mainframe, lo cual imposibilita la creación de aplicaciones distribuidas que abarquen diferentes puntos geográficos. Muchas compañías tienen empleados en todo el mundo y muchos de ellos necesitan crear sistemas para clientes y proveedores que nunca podrían integrarse en un entorno tan controlado como una red de área local (LAN) corporativa [15].
- Los usuarios están restringidos a plataformas determinadas. Por ejemplo, un sistema de dos capas suele requerir que todo el mundo ejecute el mismo sistema operativo, como, por ejemplo, una versión de 32 bits de Windows. En la situación actual en la que cualquier equipo puede conectarse a otro, esta limitación resulta cada vez más inadmisible [15].
- Los sistemas de dos capas dejan de funcionar cuando se desconectan los equipos como, por ejemplo, cuando el servidor de bases de datos se desconecta para el mantenimiento o los usuarios de equipos portátiles se desconectan de la red mientras trabajan durante un viaje o en sitios de clientes. La disponibilidad del sistema se ve limitada en gran medida si experimenta un período de inactividad cada vez que varios equipos no puedan conectarse directamente entre sí [15].

División de la capa de presentación

Para resolver los problemas que surgen de una arquitectura de dos capas se pueden desacoplar las aplicaciones cliente del código de la lógica de negocio y de acceso a datos mediante la introducción de un conjunto de objetos de negocio, tal y como se muestra en la Figura 11. La aplicación cliente que contiene el código de interfaz de usuario se suele denominar *capa de presentación*. El término *aplicación cliente* hace referencia a una aplicación de la capa de presentación que contiene los elementos y el código para la interfaz de usuario.

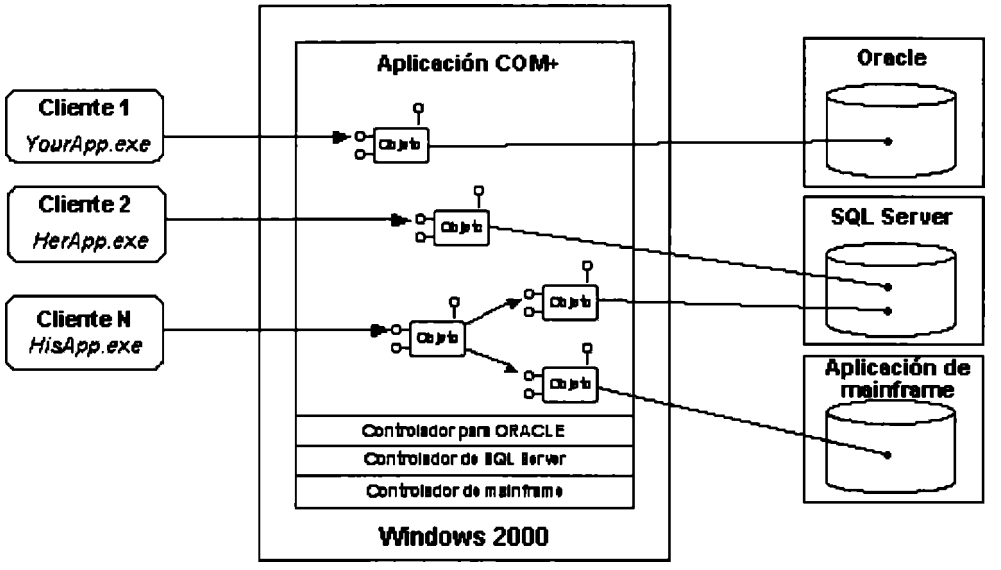


Figura 11. La introducción de un conjunto de objetos de negocio en la capa media elimina las costosas dependencias entre las aplicaciones cliente y los servidores de bases de datos.

Los objetos de negocio permiten centralizar y volver a utilizar su lógica en diversas aplicaciones cliente. En redes basadas en Windows 2000 y Windows NT® estos objetos de negocio pueden instalarse utilizando COM como base. COM también proporciona la base que permite la comunicación remota entre las aplicaciones cliente y los objetos de las capas medias [15].

Uno de los mayores atractivos comerciales de COM consiste en que permite que los programadores de capas medias actualicen el código en sus objetos de negocio sin tener que volver a compilar o distribuir las aplicaciones cliente. Modificar el código que implementa la lógica de negocio y de acceso a datos después de haber puesto en producción los componentes resulta un proceso directo y sencillo, ya que las aplicaciones cliente se encuentran protegidas de los detalles de implementación de los objetos de negocio [15].

Los datos de una aplicación pueden modificar su formato de almacenamiento y los nuevos servidores de bases de datos pueden conectarse sin ningún problema. Normalmente, se pueden realizar las modificaciones necesarias en una sola biblioteca de vínculos dinámicos (DLL) de Visual Basic®. Es fácil volver a compilar una DLL y sustituirla en un servidor de producción en la capa media. Las aplicaciones cliente permanecen sin cambios durante la producción.

En una aplicación de n-capas típica como la que se muestra en la Figura 11, se ejecuta un conjunto de objetos en la capa de negocio en nombre de varios usuarios diferentes. Sin embargo, es frecuente que todos estos objetos se ejecuten dentro de un único proceso. Esto permite que distintos usuarios puedan compartir recursos específicos del proceso como threads, memoria y conexiones a bases de datos. Asimismo, los equipos que ejecutan aplicaciones cliente no necesitan controladores de bases de datos, como solía ocurrir en el modelo de dos capas [15].

Instalación física de las capas de negocio y datos

Una de las primeras preguntas que surgen durante el desarrollo de n-capas plantea dónde deberían instalarse cada una de las capas. La Figura 12 muestra dos casos posibles. El número de capas no indica necesariamente cuantos equipos están incluidos. En una instalación pequeña, la lógica de negocio y el servidor de bases de datos pueden ejecutarse en el mismo equipo. En un sistema de mayor tamaño, los datos pueden almacenarse en uno o varios servidores específicos de bases de datos mientras que los objetos de negocio se ejecutan en un host independiente.

Algunos esquemas de instalación física resultan menos costosos y más fáciles de configurar. Otros ofrecen niveles más elevados de escalabilidad, confiabilidad y tolerancia a errores. Cuando se habla de sistemas de n-capas, se hace referencia a varias *capas lógicas* distintivas. La instalación física puede variar y modificarse fácilmente después de que un sistema se haya puesto en producción. Una de las ventajas de COM es que permite realizar modificaciones en la instalación física sin tener que cambiar o volver a compilar ningún código de la aplicación.

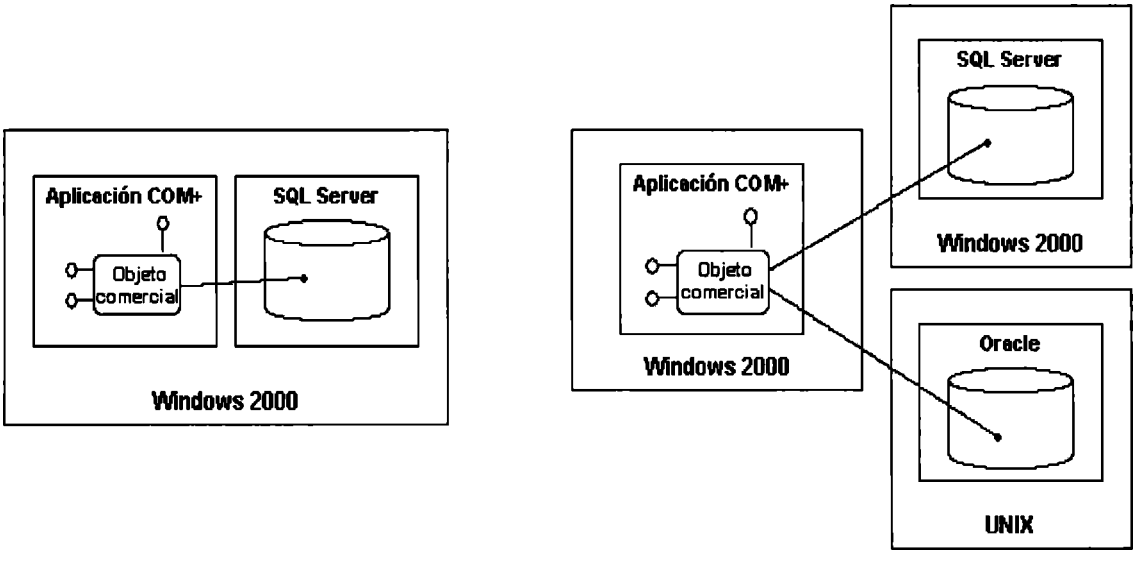


Figura 12. La asignación entre objetos de negocio y la capa de acceso a datos puede resultar sencilla al principio y volverse más compleja sin afectar a las aplicaciones cliente.

Los términos varias capas, tres capas o n-capas, hacen referencia al mismo concepto, es decir, que un sistema dispone de tres o más capas. La Figura 13 muestra un

conjunto más complejo de capas físicas. En el modelo de n-capas, las capas de la lógica de negocio y de acceso a datos pueden resultar arbitrariamente complejas. Lo mejor de este modelo es que las aplicaciones cliente únicamente saben de la existencia de una capa visible de objetos de negocio. No se preocupan de toda la complejidad adicional que reside en los objetos de negocio. Por lo tanto, el objetivo principal de diseño en un sistema de n-capas consiste en ocultar, en la medida de lo posible, toda esta complejidad en las aplicaciones cliente que forman la capa de presentación.

Otra función efectiva de COM es que permite que las aplicaciones cliente creen y utilicen objetos de toda la red. Detrás del escenario, COM utiliza un protocolo denominado *invocación a procedimiento remoto* (RPC) para ejecutar llamadas a métodos a través de los límites del host y del proceso. RPC es el primero de una serie de protocolos importantes que se debería conocer antes de diseñar una aplicación distribuida con COM+ [15].

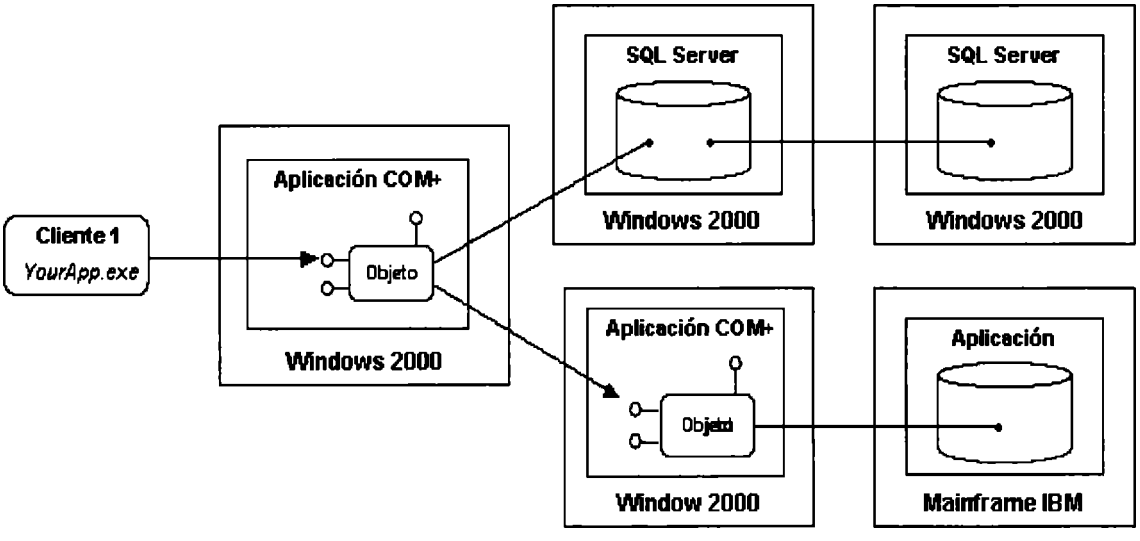


Figura 13. Una de las ventajas más importantes de una arquitectura de varias capas consiste en ocultar la complejidad de la infraestructura de TI en las aplicaciones cliente.

Si se dispone de un grupo de usuarios en el que todos ejecutan sistemas operativos compatibles con COM como lo son las versiones actuales de la familia de sistemas operativos de Microsoft Windows, se pueden crear aplicaciones de varias capas como las que se muestran en la figura anterior. Los equipos que ejecutan aplicaciones cliente permiten que COM establezca las conexiones con objetos de negocio a través de la red de área local (LAN). Una vez que una aplicación cliente crea objetos de negocio en toda la red, puede utilizarlos para ejecutar transacciones y recuperar información.

Mientras que este enfoque funciona correctamente para algunas aplicaciones, resulta inadecuado para otras. Este estilo de desarrollo suele exigir que cada usuario ejecute una versión de Windows. Asimismo, funciona con mayor confiabilidad cuando todas las aplicaciones cliente y los objetos de negocio se ejecutan en la misma red de área local (LAN). Sin embargo, puede que se desee llegar a usuarios que no se encuentren conectados a la red local o que ejecutan otros sistemas operativos como Macintosh, OS2 o UNIX, para lo cual es necesario utilizar una estrategia de desarrollo basada en Web.

Aplicaciones basadas en Web

La popularidad de Internet y de las aplicaciones basadas en Web ha fomentado la adopción por parte de la industria de las arquitecturas de n-capas. En un sistema basado en Web, las aplicaciones cliente se ejecutan dentro de un explorador de Internet (o browser). Los browsers envían peticiones a los servidores Web a través del protocolo estándar de Internet denominado *Protocolo de transferencia de hipertexto (HTTP)*. La capa de presentación en una aplicación basada en Web se construye generalmente con el *Lenguaje de marcado de hipertexto (HTML)*.

El funcionamiento de una aplicación Web típica resulta bastante sencillo. El cliente envía una petición al servidor Web y éste responde procesando la petición y enviando al usuario una página basada en HTML. La principal ventaja de utilizar HTTP y HTML es que son compatibles con las plataformas más importantes. Este tipo de aplicación podrá llegar potencialmente a cualquier usuario en Internet.

Hay que tener en cuenta que el servidor Web es el punto de entrada del cliente en la capa de negocio, como se muestra en la Figura 14. Cuando un cliente envía una solicitud HTTP, la aplicación de la capa de negocio debe cargar y ejecutar los objetos de negocio para poner de manifiesto las ventajas del desarrollo en n-capas. Mientras el código y las páginas HTML que forman la capa de presentación se sitúen en el servidor, se podrán separar los elementos de la interfaz de usuario, la lógica de negocio y la capa de acceso a datos. Se puede crear un único conjunto de componentes de negocio y compartirlos entre clientes basados en LAN y clientes basados en Web.

Si se diseña correctamente una aplicación basada en Web, se pueden comprobar todas las ventajas del desarrollo en n-capas. Se pueden compartir recursos específicos de procesos como threads, memoria y conexiones a bases de datos entre un gran número de usuarios. Se puede responder a una única solicitud HTTP recuperando datos y ejecutando transacciones contra varios orígenes de datos. Además, el desarrollo basado en Web disminuye o elimina de forma significativa los problemas de configuración del lado del cliente [15].

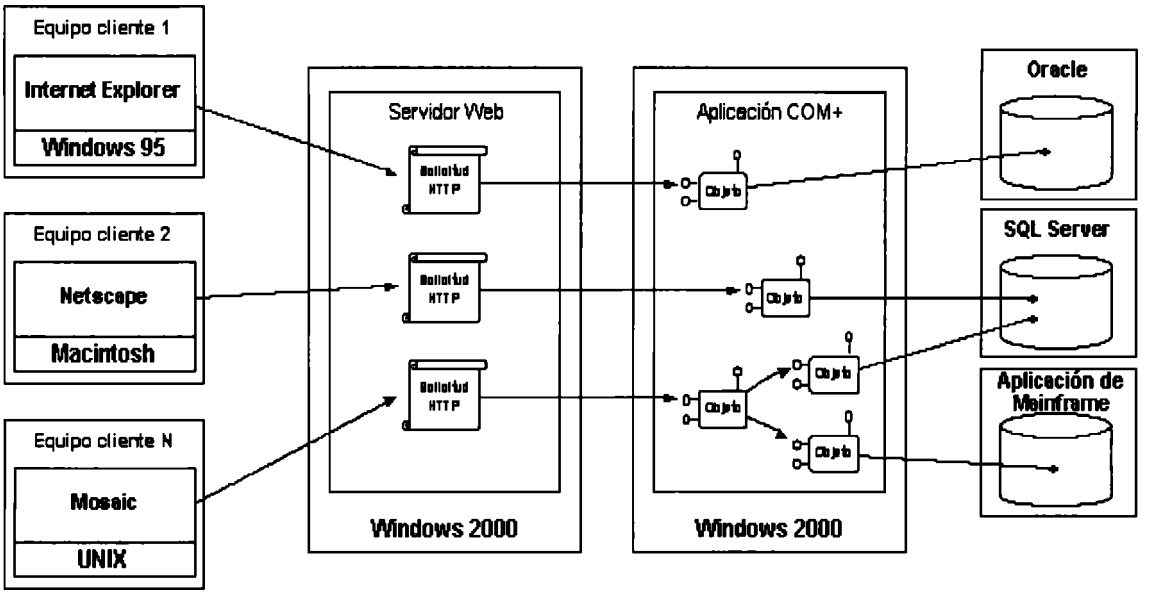


Figura 14. Con el desarrollo basado en Web se puede llegar, a través de Internet, a clientes que ejecutan una amplia gama de exploradores de Internet y sistemas operativos.

2.2. La base: COM

La estrategia de n-capas de Microsoft se basa en una tecnología central conocida como *Modelo de objetos de componentes (COM)*. Aunque COM ofrece numerosas ventajas, se trata de una tecnología compleja que incluye varios conceptos nuevos y una gran cantidad de detalles de bajo nivel. Algunos de estos conceptos y muchos de estos detalles resultan cruciales para comprender cómo crear correctamente los componentes de la capa de negocio para una aplicación COM+ [15].

El modelo de programación COM permite la distribución de los componentes de forma binaria. Esto significa que los componentes que adhieren a COM se pueden utilizar sin ninguna dependencia respecto al código fuente. Los programadores pueden enviar su trabajo en forma de archivos binarios sin revelar los algoritmos de su propiedad. La reutilización de componentes, también elimina muchos problemas en tiempo de compilación que suceden cuando las aplicaciones son armadas usando un estilo de programación basado en la reutilización del código fuente.

Antes de la aparición de las tecnologías basadas en componentes, como es el caso de COM, las aplicaciones de gran tamaño se creaban mediante el envío de cientos e incluso miles de archivos de código fuente a un compilador para generar un archivo ejecutable (un .EXE, por ejemplo). Este estilo de desarrollo, que depende de *aplicaciones monolíticas,* necesita ejecutables de gran tamaño y largos períodos de generación. Asimismo, para que tenga efecto una modificación en una sola línea de código fuente, se debe volver a crear la aplicación en su totalidad. Esto dificulta cada vez más la coordinación de los equipos de programación que trabajan de manera conjunta en una aplicación de gran tamaño. El mantenimiento y la mejora de una aplicación monolítica resultan como mínimo difícil [15].

El desarrollo basado en componentes soluciona muchos de los problemas relacionados con las aplicaciones monolíticas. Permite que los equipos de desarrollo

envíen los componentes como archivos binarios en lugar del código fuente. Los componentes binarios se pueden actualizar independientemente y sustituir en su contexto, lo que facilita el mantenimiento y la ampliación de una aplicación una vez puesta en producción. Se suele estar de acuerdo en que el uso de una tecnología basada en componentes resulta un requisito indispensable en el desarrollo de un gran sistema de información.

COM se basa en programación orientada a objetos (OOP), es decir, que permite la comunicación de clientes con objetos. COM utiliza el paradigma OOP para obtener niveles más elevados de reutilización y mantenimiento que los que se pueden alcanzar con otros modelos de reutilización binaria. Los clientes de COM y sus clases suelen ubicarse en archivos binarios independientes. COM presenta una infraestructura que permite que los clientes creen objetos y se enlacen a ellos en tiempo de ejecución.

Una plataforma basada en la reutilización de componentes orientados a objetos debe proporcionar un mecanismo dinámico de carga de clases. Este es uno de los puntos comunes entre los EJB (Enterprise Java Beans) de Java y COM. El cliente crea objetos en tiempo de ejecución nombrando un clase específica compilada en un componente binario independiente. Un agente suministrado por el sistema localiza el código de la clase, lo carga y crea el objeto solicitado.

La reutilización de componentes facilita aún más la incorporación de pequeñas modificaciones en una aplicación. Así, por ejemplo, se puede realizar la corrección de un pequeño error o una modificación del rendimiento en una *biblioteca de vínculos dinámicos* (DLL). La DLL se puede volver a compilar y sustituir en su contexto sin afectar negativamente a ninguna de las aplicaciones cliente que la utilicen. Los sistemas basados en la reutilización de código fuente deben volver a compilar todas las líneas de código de toda la aplicación, lo que dificulta y aumenta el costo de mantenimiento y la ampliación del software [15].

Con los principios de la reutilización de componentes se pueden construir aplicaciones basadas en COM utilizando componentes independientes del lenguaje. Cuando hay varios equipos de desarrollo que generan componentes para un único sistema, cada uno puede elegir su propio lenguaje de programación de manera independiente. Actualmente, la lista de lenguajes habilitados para COM incluye C++, Visual Basic, Java, Delphi e incluso COBOL[26]. Cada equipo puede optar por el lenguaje que se adapte a su experiencia de programación y que sea capaz de proporcionar la mejor combinación de flexibilidad, rendimiento y productividad.

Por ejemplo, si un equipo necesita código para sistemas de bajo nivel puede utilizar C++ gracias a la flexibilidad que ofrece. Otro equipo que escriba y amplíe el código que implementa la lógica de negocio y de acceso a datos para la misma aplicación podrá utilizar Visual Basic .NET por sus altos niveles de productividad. La combinación y selección de lenguajes permite que las compañías aprovechen al máximo el talento de sus equipos de programación.

Programación basada en interfaces

COM presenta una arquitectura que se basa en la *programación basada en interfaces*. Este estilo de programación ya había sido adoptado con anterioridad por

académicos universitarios de la rama informática y por organizaciones que necesitaban generar aplicaciones escalables y de grandes dimensiones. Este tipo de programación fue iniciada en lenguajes como C++ y Smalltalk, que no disponen de soporte formal para el concepto de interfase definida y autónoma. Hoy en día, los lenguajes y herramientas como Java y Visual Basic .NET cuentan con un soporte integrado para este estilo de programación.

Una interfase, tal como una clase, es un tipo de datos distintivo, y define un conjunto de métodos públicos sin incluir su implementación. En otras palabras, una interfase define un protocolo muy específico para la comunicación que se produce entre un cliente y un objeto. El hecho de separar la interfase de la clase o clases que la implementan permite que los creadores de clases puedan realizar actividades que de otro modo no se hubiesen podido llevar a cabo.

2.3. COM distribuido (DCOM)

Desde el principio, COM se diseñó para ir más allá de los límites de un proceso. Las primeras versiones de COM lo hacían posible únicamente cuando el proceso cliente y el proceso servidor se ejecutaban en el mismo equipo. Con el lanzamiento de Microsoft Windows NT 4.0, se agregó soporte para COM que permitió extender esta comunicación entre procesos más allá de los límites de un equipo, así, se creó un nuevo protocolo de conexión para COM que permitió el uso de aplicaciones distribuidas en un entorno de red local (LAN).

Como fue mencionado anteriormente, el soporte de COM para las aplicaciones distribuidas se basa en un mecanismo entre procesos denominado *invocación a procedimiento remoto* (RPC). RPC es un estándar industrial que ha madurado en diferentes plataformas. Se mejoró RPC con extensiones orientadas a objetos con el fin de albergar a COM; la implementación resultante en la plataforma de Windows se conoce como *Object RPC (ORPC)* [15].

COM y RPC mantienen una relación simbiótica. COM proporciona a RPC un entorno orientado a objetos y RPC ofrece a COM la capacidad de mostrar objetos en toda la red [15]. COM se sirve de RPC para realizar llamadas, de forma transparente, a métodos de equipos remotos, mediante el envío de paquetes de solicitud y respuesta entre clientes y objetos; los clientes invocan los métodos de la misma manera como si los objetos estuvieran cercanos.

2.4. MTS

Microsoft Transaction Server (MTS) es un componente de software creado para Windows NT Server. MTS permite que los objetos de la capa de negocio ejecuten y controlen transacciones distribuidas. Sin embargo, es mucho más que un administrador de transacciones; proporciona un entorno en tiempo de ejecución totalmente nuevo para los objetos COM que se ejecutan en la capa de negocio. En MTS se agregó soporte para la infraestructura que no estaban incluidas en COM, en concreto, se agregó un nuevo

soporte para las transacciones distribuidas, seguridad integrada, pooling de threads y mejoras en la configuración y administración [15].

El término MTS causó confusión, debido a que el software no se creó únicamente para compañías que tenían por objetivo ejecutar transacciones distribuidas, sino que proveyó a Microsoft del vehículo para arribar a la siguiente generación de su entorno para aplicaciones distribuidas. Los objetos de la capa de negocio destinados a Windows NT Server debían ejecutarse en el entorno MTS, intervinieran o no en las transacciones. Los usuarios suelen confundirse al saber que MTS se puede utilizar para objetos que no intervienen en transacciones. Para eliminar dicha confusión, posteriormente en Windows 2000, se cambió el nombre del entorno de ejecución de la capa de negocio de MTS a COM+ [4]. La compatibilidad con transacciones creadas para MTS se incorpora también en COM+.

MTS consiste de varios servicios combinados en un solo paquete:

- Un servidor de componentes que alberga objetos COM.
- Un monitor de transacciones para el entorno de Windows que provee procesamiento automático de transacciones.
- Un administrador de recursos que realiza pooling de conexiones de objetos, pooling de threads y activación just-in-time.

MTS y COM+ cuentan con un explorador que permite realizar tareas de configuración y administración, tal como se muestra en la figura siguiente.

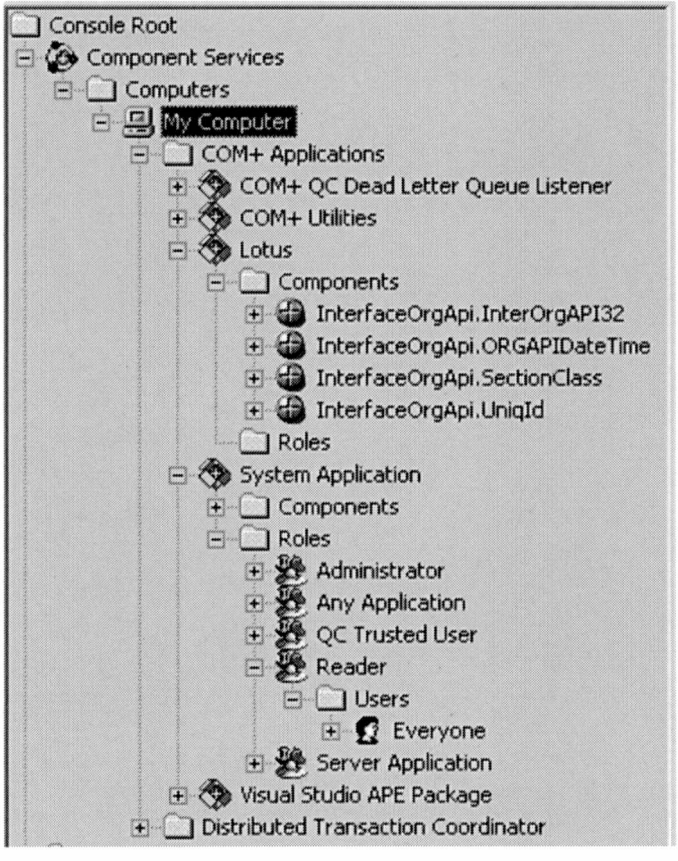


Figura 15. Explorador de COM+ (Component Services)

MTS amplió el modelo de seguridad de COM. La seguridad en MTS se basa en la noción de *roles*. Un rol es una abstracción que representa un perfil de seguridad para uno o varios usuarios de una aplicación MTS. En el momento del diseño, el programador puede configurar los controles de seguridad utilizando roles de forma declarativa o mediante programación. Durante la distribución, el administrador asigna un conjunto de roles a las cuentas de usuario y de grupo dentro de un dominio de Windows. El modelo de seguridad basado en roles proporciona una mayor flexibilidad y granularidad que el modelo de seguridad proporcionado por COM originalmente [15] [4].

Otra característica importante que MTS agrega a la plataforma es un esquema que administra la concurrencia controlando el pooling de threads de forma oculta. MTS introdujo una abstracción denominada *actividad*, que representa un thread lógico de ejecución en una aplicación MTS. Los programadores de MTS deben escribir sus aplicaciones para proporcionar una actividad por aplicación cliente. El run-time de MTS enlaza automáticamente las actividades lógicas con los threads físicos. Una vez que el número de clientes alcanza el umbral predefinido, el run-time de MTS comienza a compartir threads individuales entre varios clientes [15].

Las aplicaciones de n-capas que utilizan un esquema de pooling de threads evolucionan mejor que las que utilizan un modelo con un único thread o un modelo con un thread por cliente. El modelo con un único thread anula cualquier posibilidad de concurrencia, ya que no puede ejecutar métodos para dos o más clientes al mismo tiempo. El modelo con un thread por cliente provoca un uso inadecuado de recursos en aplicaciones de gran escala, debido a la constante necesidad de crear y destruir threads físicos. El objetivo que se persigue con un esquema de pooling de threads, como el que se generó en MTS, es el de crear un equilibrio optimizado entre niveles más elevados de concurrencia y un uso más eficiente de recursos [15].

Otra función importante de MTS fue la introducción de un soporte mejorado para la configuración del equipo y la administración de redes. Con COM, se hacía difícil y costosa la distribución y administración de aplicaciones de gran tamaño basadas en COM en un entorno de red. Las herramientas de administración de MTS facilitaron bastante la configuración y administración de los servidores que ejecutan objetos de la capa de negocio. Al contrario que COM, con MTS se pueden administrar varios equipos servidores desde un único escritorio.

La programación basada en atributos

MTS introdujo un concepto muy importante en el modelo de programación de la plataforma: la *programación basada en atributos*. Los servicios que proporciona la plataforma se exponen a través de atributos declarativos. La configuración de estos atributos viene determinada por los programadores en tiempo de diseño y los administradores pueden volver a configurarla después de que se haya puesto en producción una aplicación [15]. La programación basada en atributos es muy diferente de la forma en que la mayoría de los sistemas operativos han expuesto los servicios del sistema anteriormente.

Tradicionalmente, los sistemas operativos han expuesto los servicios a través de un conjunto de funciones en una *interfase de programación de aplicaciones* (API). En este modelo, una aplicación se sirve de los servicios del sistema haciendo llamadas

explícitas a la API; lo que implica que se deben compilar dentro de la aplicación las llamadas explícitas a nivel de sistema. Si se desea cambiar la forma de utilización de un servicio del sistema después de que una aplicación esté en producción, entonces se debe modificar el código y volver a compilar la aplicación. El modelo de programación basado en atributos declarativos resulta mucho más flexible.

Para clarificar el funcionamiento de los atributos declarativos, tomemos el siguiente ejemplo: un componente de una aplicación MTS dispone de un atributo que le indica al entorno de ejecución MTS si dicho componente admite transacciones. Si el programador especifica que un componente necesita una transacción, las instancias de los objetos del componente se crean dentro del contexto de una transacción lógica de MTS. Cuando el cliente llama a uno de estos objetos transaccionales, el run-time de MTS realiza automáticamente una llamada para comenzar una transacción física. Además, realiza la llamada correspondiente para realizar (commit) o detener (roll back) la transacción. La cuestión es que los programadores nunca deben realizar llamadas explícitas para comenzar, detener o anular una transacción. Todas las llamadas necesarias se realizan de forma oculta en el run-time de MTS.

¿Por qué es mejor el modelo de programación basado en atributos que el modelo procedural antiguo que requiere llamadas explícitas a la API? En primer lugar, el nuevo modelo necesita menos código. Las preferencias se indican durante el tiempo de diseño configurando los atributos y el entorno de ejecución de MTS asegura que esas preferencias sean alcanzadas. En segundo lugar, una razón que no resulta tan obvia es que los administradores pueden reconfigurar fácilmente la forma en que una aplicación utiliza un servicio del sistema después de que se haya puesto en producción. No es necesario modificar ningún código ni volver a compilar la aplicación.

El modelo de programación basado en atributos de MTS se basa en el mecanismo conocido como *interceptación*. Cuando el cliente crea un objeto desde un componente que ha sido configurado en una aplicación MTS, el entorno de ejecución incorpora una *capa de interceptación* tal como se muestra en la Figura 16. Esta capa representa un enlace que permite que el run-time de MTS realice procesamiento previo y posterior en las llamadas a métodos. El sistema toma el control justo antes y después de que un objeto ejecute el código de cada método.

Figura 16. Un modelo de programación basado en atributos cuenta con la intercepción para servirse de los servicios del sistema.

El conflicto de COM frente a MTS

Uno de los dilemas más complejos para los programadores que generan aplicaciones de n-capas destinados a Windows NT Server consiste en decidir cuándo utilizar COM frente a MTS. Muchas compañías han optado por MTS porque ofrece valiosos servicios integrados que COM no proporciona. Otras compañías se han

decidido por COM porque no aprecian el valor agregado del entorno en tiempo de ejecución de MTS.

MTS es simplemente una capa generada sobre COM. COM nunca se ha modificado para adaptarse a MTS. En consecuencia, las dos capas en tiempo de ejecución no están totalmente integradas como podrían estarlo. Como resultado tenemos una plataforma que dispone de cierto grado de ineficacia, ambigüedad y confusión. La situación se vuelve difícil para los programadores de MTS, ya que éstos necesitan conocer el funcionamiento de ambos modelos de programación. Además, varias técnicas válidas de programación de COM no funcionan adecuadamente cuando se utilizan en una aplicación MTS.

2.5. De COM y MTS a COM+

Con el lanzamiento de Windows 2000, las mejores características de COM y MTS se integraron en un nuevo entorno de ejecución denominado *COM+*. Al contrario que lo que ocurría con MTS, este nuevo entorno de ejecución no es opcional, es decir, COM+ forma parte de la instalación predeterminada de Windows 2000. Además, el dilema COM frente a MTS ya no existe, incluso, la escritura de los componentes para COM+ resulta más sencilla que la de MTS.

Al igual que COM, COM+ se basa en componentes binarios y en la programación basada en interfaces. Las llamadas a métodos se realizan de forma remota a través de los límites del equipo y del proceso mediante el uso de una capa transparente de RPC. Al igual que los componentes COM, los componentes COM+ se pueden actualizar y ampliar en producción sin que se vean afectadas de forma negativa las aplicaciones de cliente que los utilizan.

Del mismo modo que MTS, COM+ admite transacciones distribuidas y un sistema de seguridad basado en roles. Proporciona, además, un esquema de pooling de threads igual de transparente que el de MTS. El modelo de programación COM+ también utiliza la *capa de interceptación* para mostrar a los programadores servicios de plataforma a través de atributos declarativos. Sin embargo, COM+ adopta la programación basada en atributos con mayor profundidad que MTS. Además de los servicios transaccionales y el sistema de seguridad integrada, COM+ ofrece servicios como la construcción de objetos personalizada, la sincronización y el pooling de objetos. Otras nuevas funciones de COM+, como los componentes encolados y los eventos COM+ se muestran también a través de atributos configurables [15].

Este es un breve sumario de los servicios provistos por COM+ [13]:

- *Administración*. Herramientas y servicios que permiten a los programadores y administradores configurar y manejar componentes y aplicaciones basadas en componentes.
- *Activación Just-In-Time (JITA)*. Servicios que instancian componentes cuando ellos son llamados y son desechados cuando terminan de hacer su trabajo.
- *Pooling de objetos*. Servicios que permiten que instancias usadas frecuentemente, pero costosas en cuanto a recursos, tales como conexiones a base de datos, sean mantenidas en un pool para que sean usados por los clientes.

El pooling de objetos puede mejorar la performance y los tiempos de respuesta de aplicaciones distribuidas.

- *Transacciones*. Servicios que permiten que varias operaciones sean tratadas como una única operación. El manejo de transacciones es un requerimiento de la mayoría de los sistemas comerciales.
- *Sincronización*. Servicios para controlar el acceso concurrente a los objetos.
- *Seguridad*. Servicios para autenticación de clientes y control de acceso a una aplicación. COM+ soporta seguridad basada en roles.
- *Componentes encolados*. Servicios que permiten que los componentes se comuniquen a través de mensajes asincrónicos.
- *Eventos*. Servicios que permiten que los componentes se informen unos a otros de eventos significativos tales como cambios en los datos. COM+ soporta el modelo de notificación de eventos *publisher-suscriber*.

Componentes Configurados COM+

COM+ permite importar sólo componentes *in-proc* (DLL). No se pueden importar componentes COM que residan en un servidor local (EXE). COM+ permite configurar el tipo de activación de la aplicación, *librería o servidor*. En el caso de una librería, el cliente simplemente carga la DLL original en su proceso y usa el componente. Si se configura la aplicación para que sea una aplicación de servidor, COM+ promueve su DLL original para hostearla en un proceso dependiente del suyo propio. Sin embargo, COM+ no puede hacer una aplicación de librería fuera de un servidor local COM [13].

Una vez que una DLL es importada en COM+, es llamada *componente configurado* para enfatizar el hecho de que la funcionalidad y el comportamiento del componente es actualmente configurado y administrado fuera del componente. Un componente COM clásico que no ha sido importado en COM+ es llamado *componente no configurado*. Componentes configurados y no configurados pueden interactuar libremente.

Los componentes configurados deben residir sobre máquinas Windows 2000 o superior.

La configuración nos permite controlar la forma en que la aplicación, componente, interfase o método se comporte bajo COM+.

Aplicaciones, DLL's y componentes

Una aplicación COM+ es un grupo lógico de componentes COM+. Los componentes usualmente comparten una aplicación si dependen unos de otros para completar sus tareas y cuando todos las completan requieren el mismo nivel de configuración de la aplicación, como la seguridad o políticas de activación [13].

Cada aplicación COM+ es identificada unívocamente por una GUID llamada *Application ID*. Nunca se usa este identificador directamente, pero COM+ lo usa internamente.

Las aplicaciones COM+ son unidades de paquetes lógicos, las dll's, sin embargo son unidades de paquetes físicos. No hay correlación entre los paquetes lógicos y los paquetes físicos, el único requisito es que los componentes configurables deben pertenecer exactamente a una única aplicación COM+, estos no pueden pertenecer a más de una y deben pertenecer al menos a una para que tomen ventaja de los servicios. Como se ve en la siguiente figura, una aplicación COM+ puede tener componentes de una o más dll's. También es posible que no todos los componentes de una dll estén en una aplicación COM+ (como el componente E) y una dll puede contribuir con componentes a múltiples aplicaciones COM+ (la DLL 1 contribuye con componentes a las aplicaciones 1 y 2).

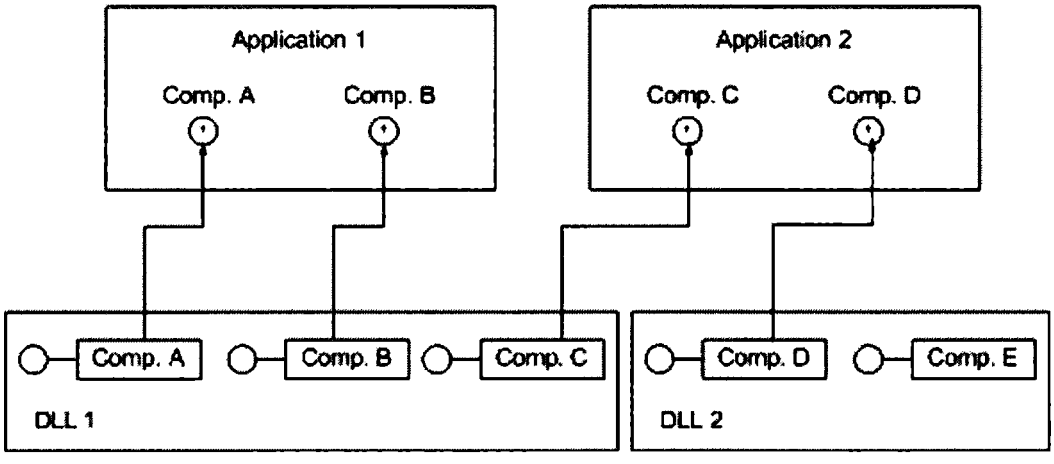


Figura 17. Aplicaciones, DLL's y componentes.

La separación del empaquetado físico del lógico da gran flexibilidad en el diseño de la disposición de la aplicación. Todos los componentes de la misma aplicación COM+ comparten los mismos seteos de configuración a nivel de aplicación. Sin embargo, se recomienda no instalar componentes de la misma dll en más de una aplicación. La razón es que los componentes en la misma aplicación COM+ se asume que operan estrechamente juntos. Por otro lado, nada es asumido sobre componentes de diferentes aplicaciones. Ubicar componentes de la misma dll en múltiples aplicaciones, puede introducir chequeos de seguridad innecesarios y una sobrecarga entre procesos si estos componentes necesitan operar entre ellos [13].

Como ejemplo de aplicación COM+ puede mencionarse aquella destinada a las operaciones sobre la BBDD. En este caso, se cuenta con una aplicación COM+ llamada *AppBBDD* que contiene tres componentes: *DataServices*, *General* y *Transaction*. Estos se encuentran dentro de una misma DLL llamada BaseDatos.dll. Dentro del componente DataServices se encuentran dos métodos utilizados para ejecutar consultas contra la BBDD. Todo esto se muestra en la figura siguiente.

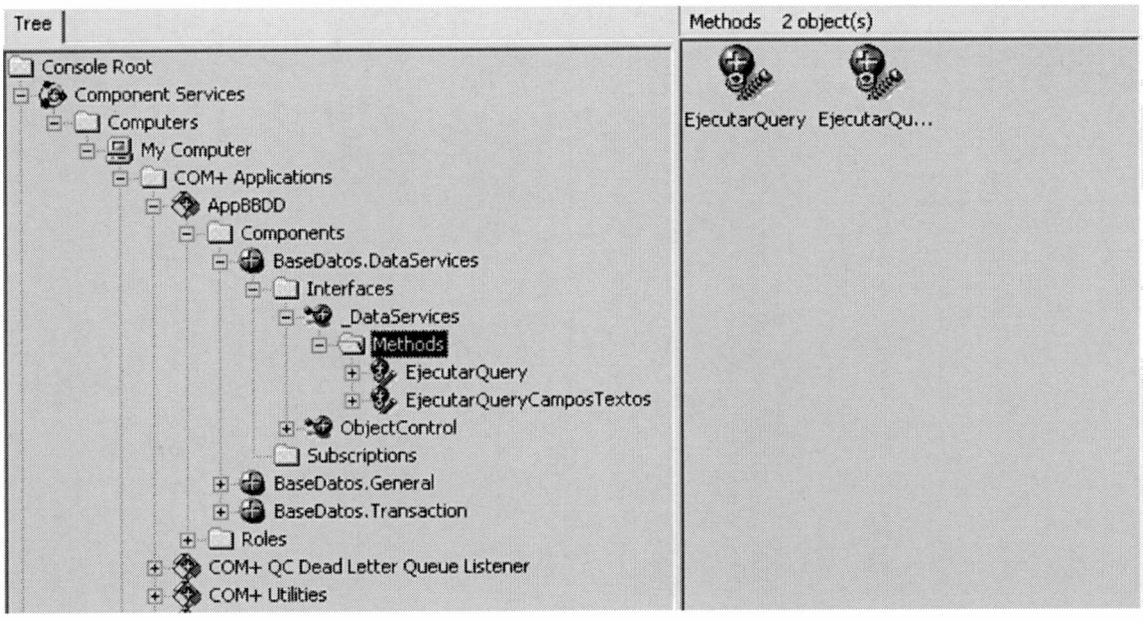


Figura 18. Aplicación COM+ desde el explorador de COM+.

3. Plataforma .NET

3.1. Microsoft .NET

Microsoft .NET es el conjunto de nuevas tecnologías en las que Microsoft ha estado trabajando durante los últimos años con el objetivo de obtener una plataforma sencilla y potente para distribuir el software en forma de servicios que puedan ser suministrados remotamente y que puedan comunicarse y combinarse unos con otros de manera totalmente independiente de la plataforma, lenguaje de programación y modelo de componentes con los que hayan sido desarrollados. Ésta es la llamada **plataforma .NET**, y a los servicios antes comentados se les denomina **servicios Web**.

Para crear aplicaciones para la plataforma .NET, tanto servicios Web como aplicaciones tradicionales (aplicaciones de línea de comando, aplicaciones Windows, etc.), Microsoft ha publicado el denominado kit de desarrollo de software conocido como **.NET Framework SDK**, que incluye las herramientas necesarias tanto para su desarrollo como para su distribución y ejecución. Entre ellas contamos con los distintos compiladores para los distintos lenguajes (C#, Visual Basic .NET, J# y C++ con extensiones gestionadas), una herramienta de ingeniería inversa que reconstruye para poder examinar el código fuente de un archivo compilado para la plataforma (Intermediate Language Disassembler, ildasm.exe), etc. También nos provee del **Visual Studio .NET**, que es el IDE para desarrollo en los distintos lenguajes mencionados anteriormente.

El concepto de Microsoft .NET también incluye al conjunto de nuevas aplicaciones que Microsoft y terceros han (o están) desarrollando para ser utilizadas en la plataforma .NET. Entre ellas podemos destacar aplicaciones desarrolladas por Microsoft tales como Windows Vista, Visual Studio .NET, MSN.NET, Office.NET, y los nuevos servidores para empresas de Microsoft (SQL Server 2005, Exchange.NET, etc.).

Common Language Runtime (CLR)

El **Common Language Runtime (CLR)** es el núcleo de la plataforma .NET. Es el motor encargado de gestionar la ejecución de aplicaciones. Les que ofrece numerosos servicios que simplifican su desarrollo y favorecen su fiabilidad y seguridad. Las principales características y servicios que ofrece el CLR son: [18]

- **Modelo de programación consistente:** a todos los servicios y facilidades ofrecidos por el CLR se accede: a través de un modelo de programación orientado a objetos. Todos son objetos. Esto es una diferencia importante respecto al modo de acceso a los servicios ofrecidos por algunos sistemas operativos Windows, en los que por ejemplo a los servicios se les accede a

través de llamadas a funciones globales definidas en DLLs y a otros a través de objetos (objetos COM).

- **Modelo de programación sencillo:** con el CLR desaparecen muchos elementos complejos incluidos en los sistemas operativos actuales (registro de Windows, GUIDs, etc.) El CLR no es que abstraiga al programador de estos conceptos, sino que son conceptos que no existen en la plataforma .NET.

- **Eliminación del "infierno de las DLLs":** en la plataforma .NET desaparece el problema conocido como "infierno de las DLLs" que se da en los sistemas operativos actuales de la familia Windows, problema que consiste en que al sustituirse versiones viejas de DLLs compartidas por versiones nuevas puede que aplicaciones que fueron diseñadas para ser ejecutadas usando las viejas dejen de funcionar si las nuevas no son 100% compatibles con las anteriores. En la plataforma .NET las versiones nuevas de las DLLs pueden coexistir con las viejas, de modo que las aplicaciones diseñadas para ejecutarse usando las viejas podrán seguir usándolas tras instalación de las nuevas. Esto, obviamente, simplifica mucho la instalación y desinstalación de software.

- **Ejecución multiplataforma**: el CLR actúa como una máquina virtual, encargándose de ejecutar las aplicaciones diseñadas para la plataforma .NET. Es decir, cualquier sistema operativo para la que exista una versión del CLR podrá ejecutar cualquier aplicación .NET. Microsoft ha desarrollado versiones del CLR para la mayoría de las versiones de Windows. También hay terceros que están desarrollando de manera independiente versiones de distribución libre del CLR para Linux (*ver Proyecto Mono, pág. 37*).

- **Integración de lenguajes:** desde cualquier lenguaje para el que exista un compilador que genere código para la plataforma .NET es posible utilizar el código generado por cualquier otro lenguaje para el que también exista un compilador que genere código para la plataforma .NET. Microsoft ha desarrollado un compilador de C# que genera código de este tipo, así como Visual Basic.NET, C++ con extensiones gestionadas y Jscript.NET que también lo generan. Con esta integración de lenguajes es posible escribir una clase C# que herede de otra escrita en Visual Basic.NET que, a su vez, herede de otra escrita en C++ con extensiones gestionadas.

- **Gestión de memoria:** el CLR incluye un **recolector de basura** que evita que el programador tenga que tener en cuenta cuando ha de destruir los objetos que dejen de serle útiles. Este recolector es una aplicación que se activa cuando se quiere crear algún objeto nuevo y se detecta que no queda memoria libre para hacerlo, caso en que el recolector recorre la memoria dinámica asociada a la aplicación, detecta qué objetos hay en ella que no son accedidos por el código de la aplicación (referenciados por otros objetos), y los elimina para limpiar la memoria de "objetos basura" y permitir la creación de otros nuevos. Gracias a este recolector se evitan errores de programación muy comunes como intentos de borrado de objetos ya borrados, agotamiento de memoria por olvido de eliminación de objetos inútiles o solicitud de acceso a miembros de objetos ya destruidos.

- **Seguridad de tipos:** el CLR facilita la detección de errores de programación difíciles de localizar comprobando que toda conversión de tipos que se realice

durante la ejecución de una aplicación .NET se haga de modo que los tipos origen y destino sean compatibles.

- **Aislamiento de procesos:** el CLR asegura que el código perteneciente a un determinado proceso no pueda acceder a código o datos pertenecientes a otro, lo que evita errores de programación muy frecuentes e impide que algunos procesos puedan atacar a otros. Esto se consigue gracias al sistema de seguridad de tipos antes comentado, pues evita que se pueda convertir un objeto a un tipo de mayor tamaño que el suyo propio, ya que al tratarlo como a un objeto de mayor tamaño podría accederse a espacios de memoria ajenos a él que podrían pertenecer a otro proceso. También se consigue gracias a que no se permite acceder a posiciones arbitrarias de memoria.

- **Tratamiento de excepciones:** en el CLR todos los errores que se puedan producir durante la ejecución de una aplicación se propagan de igual manera: mediante excepciones. Esto es muy diferente a como se venía haciendo en los sistemas Windows hasta la aparición de la plataforma .NET, donde ciertos errores se transmitían mediante códigos de error en formato Win32, otros mediante HRESULTs y otros mediante excepciones.

  El CLR permite que excepciones lanzadas desde código para .NET escrito en un cierto lenguaje se puedan capturar en código escrito usando otro lenguaje, e incluye mecanismos de depuración que pueden saltar desde código escrito para .NET en un determinado lenguaje a código escrito en cualquier otro. Por ejemplo, se puede recorrer la pila de llamadas de una excepción aunque ésta incluya métodos definidos en otros módulos usando otros lenguajes.

- **Soporte multihilo:** el CLR es capaz de trabajar con aplicaciones divididas en múltiples hilos de ejecución que pueden ir evolucionando por separado en paralelo e intercalándose, según el número de procesadores de la máquina sobre la que se ejecuten. Las aplicaciones pueden lanzar nuevos hilos, destruirlos, suspenderlos por un tiempo o hasta que les llegue una notificación, enviarles notificaciones, sincronizarlos, etc.

- **Distribución transparente:** el CLR ofrece la infraestructura necesaria para crear objetos remotos y acceder a ellos de manera completamente transparente a su localización real, tal y como si se encontrasen en la máquina que los utiliza.

- **Seguridad avanzada**: el CLR proporciona mecanismos para restringir la ejecución de ciertos códigos o los permisos asignados a los mismos según su procedencia o el usuario que los ejecute. Es decir, puede no darse el mismo nivel de confianza a código procedente de Internet que a código instalado localmente o procedente de una red local; pueden no darse los mismo permisos a código procedente de un determinado fabricante que a código de otro; y pueden no darse los mismos permisos a un mismo código según el usuario que lo esté ejecutando o según el rol que éste desempeñe. Esto permite asegurar al administrador de un sistema que el código que se esté ejecutando no pueda poner en peligro la integridad de sus archivos, la del registro de Windows, etc.

- **Interoperabilidad con código antiguo:** el CLR incorpora los mecanismos necesarios para poder acceder desde código escrito para la plataforma .NET a código escrito previamente a la aparición de la misma y, por lo tanto, no preparado para ser ejecutado dentro de ella. Estos mecanismos permiten tanto el

acceso a objetos COM como el acceso a funciones sueltas de DLLs preexistentes (como la API Win32).

Como se puede deducir de las características comentadas, el CLR lo que hace es encargarse de la ejecución de las aplicaciones diseñadas para la plataforma .NET. Por esta razón, al código de estas aplicaciones se le suele llamar **código gestionado**, y al código no escrito para ser ejecutado directamente en la plataforma .NET se le suele llamar **código no gestionado**.

Microsoft Intermediate Language (MSIL)

Todos los compiladores que generan código para la plataforma .NET no lo hacen para ningún tipo de CPU concreta, sino que generan código escrito en el lenguaje intermedio conocido como Microsoft Intermediate Language (MSIL). El CLR interpreta y ejecuta MSIL. Es decir, MSIL es el único código que es capaz de interpretar el CLR, y por tanto cuando se dice que un compilador genera código para la plataforma .NET lo que se está diciendo es que genera código MSIL.

MSIL ha sido creado por Microsoft. Es un lenguaje de un nivel de abstracción más alto que el de la mayoría de los códigos de máquina de las CPUs existentes, e incluye instrucciones que permiten trabajar directamente con objetos (crearlos, destruirlos, inicializarlos, llamar a métodos virtuales, etc.), tablas y excepciones (lanzarlas, capturarlas y tratarlas). [18]

Como se puede deducir, el compilador de C# compila directamente código fuente MSIL. Existen compiladores de diferentes lenguajes que generan código MSIL. En la actualidad se cuenta con compiladores para Visual Basic (Visual Basic.NET), C++ (C++ con extensiones gestionadas) y de JScript (JScript.NET). Pues bien, también hay numerosos terceros que han anunciado estar realizando versiones para la plataforma .NET de otros lenguajes como Cobol, Delphi, etc. Como podemos ver, esta estrategia es más abarcativa que la adoptada por Sun para Java, ya que su máquina virtual está orientada a un solo lenguaje. Como contrapartida, Java y su plataforma cumplen con estándares aprobados. Ambas plataformas son de acceso libre

La principal ventaja del MSIL es que permite la ejecución multiplataforma y la integración entre diferentes lenguajes independientes de la CPU y proporciona un formato común para el código de máquina generado por todos los compiladores que generen código para .NET. Sin embargo, dado que las CPUs no pueden ejecutar directamente MSIL, antes de ejecutarlo habrá que convertirlo al código nativo de la CPU sobre la que se vaya a ejecutar. De esto se encarga un componente del CLR conocido como compilador JIT (Just-In-Time) o jiiter que va convirtiendo dinámicamente el código MSIL a código nativo según sea necesario. Este jiiter se distribuye en tres versiones: [18]

- **jiiter normal:** es el que se suele usar por defecto, y compila el código MSIL a código nativo a medida que va siendo necesario, pues así se ahorra tiempo y memoria al evitarse tener que compilar innecesariamente código que nunca se ejecute. Para conseguir esto, el cargador de clases del CLR sustituye

inicialmente las llamadas a métodos de las nuevas clases que vaya cargando por llamadas a funciones auxiliares (stubs) que se encarguen de compilar el verdadero código del método. Una vez compilado, la llamada al stub es sustituida por una llamada directa al código ya compilado, con lo que posteriores llamadas al mismo no necesitarán compilación. [18]

- **jiiter económico:** funciona de forma similar al jiiter normal solo que no realiza ninguna optimización de código al compilar si no que traduce cada instrucción MSIL por su equivalente en el código máquina sobre la que se ejecute. Está especialmente pensado para ser usado en dispositivos empotrados (por ejemplo, lectores de código de barras que cuentan con handhelds con procesamiento limitado) que dispongan de poca potencia de CPU y poca memoria, pues aunque genere código más ineficiente es menor el tiempo y memoria que necesita para compilar. Es más, para ahorrar memoria este jiiter puede descargar código ya compilado que lleve cierto tiempo sin ejecutarse y sustituirlo de nuevo por el stub apropiado. Por estas razones, este es el jiiter usado por defecto en Windows CE, sistema operativo que se suele incluir en los dispositivos empotrados antes mencionados. [18]

  Otra utilidad del jiiter económico es que facilita la adaptación de la plataforma .NET a nuevos sistemas porque es mucho más sencillo de implementar que el normal.[18]

- **prejiiter:** se distribuye como una aplicación en línea de comandos llamada **ngen.exe** mediante la que es posible compilar completamente cualquier ejecutable o librería (cualquier ensamblado en general, aunque este concepto se verá más adelante) que contenga código gestionado y convertirlo a código nativo, de modo que posteriores ejecuciones del mismo se harán usando esta versión ya compilada y no se perderá tiempo en hacer la compilación dinámica. [18]

La actuación de un jiiter durante la ejecución de una aplicación gestionada puede dar la sensación de hacer que ésta se ejecute más lentamente debido a que ha de invertir tiempo en las compilaciones dinámicas. Esto es cierto, pero hay que tener en cuenta que en .NET cada código no es interpretado cada vez que se ejecuta sino que sólo es compilado la primera vez que se llama al método al que pertenece. Es más, el hecho de que la compilación se realice dinámicamente permite que el jiiter tenga acceso a mucha más información sobre la máquina en que se ejecutará la aplicación del que tendría cualquier compilador tradicional, con lo que puede optimizar el código para ella generado (por ejemplo, usando las instrucciones especiales, usando registros extra, incluyendo registro *inline*, etc.). Además, como el recolector de basura de .NET mantiene siempre compactada la memoria dinámica las reservas de memoria se harán más rápido, sobre todo en aplicaciones que no agoten la memoria y, por tanto, no necesiten de una recolección de basura.[18]

En la plataforma .NET se distinguen dos tipos de **módulos** de código compilado: **ejecutables** (extensión **.exe**) y **librerías de enlace dinámico** (extensión **.dll** generalmente). Ambos son archivos que contienen definiciones de tipos de datos, y la diferencia entre ellos es que sólo los primeros disponen de un método especial que sirve de punto de entrada a partir del que es posible ejecutar el código que contienen haciendo una llamada desde la línea de comandos del sistema operativo. A ambos tipos de módulos se les suele llamar **ejecutables portables** (PE), ya que su código puede ejecutarse en cualquiera de los diferentes sistemas operativos de la familia Windows para los que existe alguna versión del CLR. [18]

El contenido de un módulo no sólo es MSIL, sino que también consta de otras dos áreas muy importantes: la cabecera de CLR y los metadatos. [18]

- La **cabecera de CLR** es un pequeño bloque de información que indica que se trata de un módulo gestionado e indica la versión del CLR que necesita, cuál es su firma digital, cuál es su punto de entrada (si es un ejecutable), etc.
- Los **metadatos** son un conjunto de datos organizados en forma de tablas que almacenan información sobre los tipos definidos en el módulo, los miembros de éstos y sobre cuáles son los tipos externos al módulo a los que se les referencia en el módulo. Los metadatos de cada módulo los genera automáticamente el compilador al crearlo, y entre sus tablas se incluyen:

Tabla	**Descripción**
ModuleDef	Define las características del módulo. Consta de un único elemento que almacena un identificador de versión de módulo (GUID creado por el compilador) y el nombre del archivo que se dio al módulo al compilarlo (así este nombre siempre estará disponible, aunque se renombre el archivo)
TypeDef	Define las características de los tipos definidos en el módulo. De cada tipo se almacena su nombre, su tipo padre, sus modificadores de acceso y referencias a los elementos de las tablas de miembros correspondientes a sus miembros.
MethodDef	Define las características de los métodos definidos en el módulo. De cada método se guarda su nombre, firma (por cada parámetro se incluye una referencia el elemento apropiado en la tabla ParamDef), modificadores de acceso y posición del módulo donde comienza el código MSIL de su cuerpo.
ParamDef	Define las características de los parámetros definidos en el módulo. De cada parámetro se guarda su nombre y modificadores.
FieldDef	Define las características de los campos definidos en el módulo. De cada uno se almacena información sobre cuál es su nombre, tipo y modificadores.
PropertyDef	Define las características de las propiedades definidas en el módulo. De cada una se indica su nombre, tipo, modificadores y referencias a los elementos de la tabla MethodDef correspondientes a sus métodos set/get.
EventDef	Define las características de los eventos definidos en el módulo. De cada uno se indica su nombre, tipo, modificadores y referencias a los

	elementos de la tabla MethodDef correspondientes a sus métodos add/remove.
AssemblyRef	Indica cuáles son los ensamblados externos a los que se referencia en el módulo. De cada uno se indica cuál es su nombre de archivo (sin extensión), versión, idioma y marca de clave pública.
ModuleRef	Indica cuáles son los otros módulos del mismo ensamblado a los que referencia el módulo. De cada uno se indica cuál es su nombre de archivo.
TypeRef	Indica cuáles son los tipos externos a los que se referencia en el módulo. De cada uno se indica cuál es su nombre y, según donde estén definidos, una referencia a la posición adecuada en la tabla AssemblyRef o en la tabla ModuleRef.
MemberRef	Indica cuáles son los miembros definidos externamente a los que se referencia en el módulo. Estos miembros pueden ser campos, métodos, propiedades o eventos; y de cada uno de ellos se almacena información sobre su nombre y signatura, así como una referencia a la posición en la tabla TypeRef donde se almacena información relativa al tipo del que es miembro.

Nótese que el significado de los metadatos es similar al de otras tecnologías previas a la plataforma .NET como lo son los archivos IDL. Sin embargo, los metadatos tienen dos ventajas importantes sobre éstas: contiene más información y siempre se almacenan incrustados en el módulo al que describen, haciendo imposible la separación entre ambos. Además, es posible consultar los metadatos de cualquier módulo a través de las clases del espacio de nombres **System.Reflection** de la BCL como añadirles información adicional mediante **atributos**. [18]

Ensamblados

Un **ensamblado** es una agrupación lógica de uno o más módulos o archivos de recursos (archivos .GIF, .HTML, etc.) que se engloban bajo un nombre común. Un programa puede acceder a información o código almacenados en un ensamblado sin tener porqué saber cuál es el archivo en concreto donde se encuentran, por lo que los ensamblados nos permiten abstraernos de la ubicación física del código que ejecutemos o de los recursos que usemos. [18]

Todo ensamblado contiene un **manifiesto**, que son metadatos con información sobre las características del ensamblado. Este manifiesto puede almacenarse en cualquiera de los módulos que formen el ensamblado o en uno específicamente creado para ello, caso éste último necesario cuando es un **ensamblado satélite** (sólo contiene recursos). [18]

Las principales tablas incluidas en los manifiestos son las siguientes: [18]

Tabla	**Descripción**
AssemblyDef	Define las características del ensamblado. Consta de un único elemento que almacena el nombre del ensamblado sin

	extensión, versión, idioma, clave pública y tipo de algoritmo de dispersión usado para hallar los valores de dispersión de la tabla FileDef.
FileDef	Define cuáles son los archivos que forman el ensamblado. De cada uno se da su nombre y valor de dispersión. Nótese que sólo el módulo que contiene el manifiesto sabrá qué archivos forman el ensamblado, pero el resto de los archivos del mismo no sabrán si pertenecen o no a un ensamblado (no contienen metadatos que les indiquen si pertenecen a un ensamblado)
ManifestResourceDef	Define las características de los recursos incluidos en el módulo. De cada uno se indica su nombre y modificadores de acceso. Si es un recurso incrustado se indica dónde empieza dentro del PE que lo contiene, y si es un archivo independiente se indica cuál es el elemento de la tabla FileDef correspondiente a dicho archivo.
ExportedTypesDef	Indica cuáles son los tipos definidos en el ensamblado y accesibles desde afuera del mismo. Para ahorrar espacio sólo recogen los que no pertenezcan al módulo donde se incluye el manifiesto, y de cada uno se indica su nombre, la posición en la tabla FileDef del archivo donde se ha implementado y la posición en la tabla TypeDef correspondiente a si definición.
AssemblyProccesorDef	Indica en qué procesadores se puede ejecutar el ensamblado. Suele estar vacía, lo que indica que se puede ejecutar en cualquier procesador; pero si estuviese llena, cada elemento indicaría un tipo de procesador admitido según el formato de identificadores de procesador del archivo WinNT.h incluido con Visual Studio.NET.
AssemblyOSDef	Indica bajo qué sistemas operativos se puede ejecutar el ensamblado, lo que puede ser útil si contiene módulos con tipos o métodos disponibles sólo en ciertos sistemas. Suele estar vacía, lo que indica que se puede ejecutar en cualquier bajo cualquier sistema operativo; pero si estuviese llena, indicaría el identificador de cada uno de los sistemas admitidos siguiendo el formato del WinNT.h de Visual Studio.NET.

Para asegurar que no se haya alterado la información de ningún ensamblado se usa el criptosistema de clave pública RSA. Lo que se hace es calcular el código de dispersión SHA-1 del módulo que contenga el manifiesto e incluir tanto este valor cifrado con RSA (**firma digital**) como la clave pública necesaria para descifrarlo en algún lugar del módulo que se indicará en la cabecera de CLR. Cada vez que se vaya a cargar en memoria el ensamblado se calculará su valor de dispersión de nuevo y se comprobará que es igual al resultado de descifrar el original usando su clave pública. Si no fuese así se detectará que se ha adulterado su contenido.

Para asegurar también que los contenidos del resto de archivos que forman un ensamblado no hayan sido alterados lo que se hace es calcular el código de dispersión de éstos antes de cifrar el ensamblado y guardarlo en el elemento correspondiente a cada

archivo en la tabla FileDef del manifiesto. El algoritmo de cifrado usado por defecto es SHA-1, aunque en este caso también se da la posibilidad de usar MD5. En ambos casos, cada vez que se accede al archivo para acceder a un tipo o recurso se calculará de nuevo su valor de dispersión y se comprobará que coincida con el almacenado en FileDef.

Dado que las claves públicas son valores que ocupan muchos bytes, lo que se hace para evitar que los metadatos sean excesivamente grandes es no incluir en las referencias a ensamblados externos de la tabla AssemblyRef las claves públicas de dichos ensamblados, sino sólo los 64 últimos bits resultantes de aplicar un algoritmo de dispersión de dichas claves. A este valor recortado se le llama **marca de clave pública**.

Hay dos tipos de ensamblados: **ensamblados privados** y **ensamblados compartidos**. Los privados se almacenan en el mismo directorio que la aplicación que los usa y sólo puede usarlos ésta, mientras que los compartidos se almacenan en un **caché de ensamblado global** (GAC) y pueden usarlos cualquiera que haya sido compilada referenciándolos. [18]

Los compartidos han de cifrarse con RSA ya que lo que los identifica en el GAC es su nombre (sin extensión) más su clave pública, lo que permite que en el GAC puedan instalarse varios ensamblados con el mismo nombre y diferentes claves públicas. Es decir, es como si la clave pública formase parte del nombre del ensamblado, razón por la que a los ensamblados así cifrados se les llama **ensamblados de nombre fuerte**. Esta política permite resolver los conflictos derivados de que se intente instalar en un mismo equipo varios ensamblados compartidos con el mismo nombre pero procedentes de distintas empresas, pues éstas tendrán distintas claves públicas.

También para evitar problemas, en el GAC se pueden mantener múltiples versiones de un mismo ensamblado. Así, si una aplicación fue compilada usando una cierta versión de un determinado ensamblado compartido, cuando se ejecute sólo podrá hacer uso de esa versión del ensamblado y no de alguna otra más moderna que se hubiese instalado en el GAC. De esta forma se soluciona el problema del **infierno de las DLL** comentado al principio del tema.

En realidad es posible modificar tanto las políticas de búsqueda de ensamblados como la política de aceptación de ensamblados compartidos incluyendo en el directorio de instalación de la aplicación un archivo de configuración en formato XML con las nuevas reglas para las mismas. Este archivo ha de llamarse igual que el ejecutable de la aplicación pero ha de tener extensión **.cfg**.

Librería de clases base (BCL)

La librería de clases base (en inglés Base Class Library, BCL) es una librería incluida en el *.NET Framework* formada por cientos de tipos de datos que permiten acceder a los servicios ofrecidos por el CLR y a las funcionalidades más frecuentemente usadas a la hora de escribir programas. Además, a partir de estas clases prefabricadas el programador puede crear nuevas clases que mediante herencia extiendan su funcionalidad y se integren a la perfección con el resto de las clases de la BCL. Por ejemplo, implementando ciertas interfaces podemos crear nuevos tipos de colecciones

que serán tratadas exactamente igual que cualquiera de las colecciones incluidas en la BCL. [18]

Esta librería está escrita en MSIL, por lo que puede usarse desde cualquier lenguaje cuyo compilador genere MSIL. A través de las clases suministradas en ella es posible desarrollar cualquier tipo de aplicación, desde las tradicionales aplicaciones de ventanas, consola o servicios de Windows hasta los novedosos servicios Web y páginas ASP.NET. Es tal la riqueza de servicios que ofrece que pueden crearse lenguajes que carezcan de librería de clases propia y sólo usen la BCL, un ejemplo es C#.

Dado la amplitud de la BCL, ha sido necesario organizar las clases en ella incluida en **espacios de nombres** que agrupen clases con funcionalidades similares. Por ejemplo, los espacios de nombres más usados son: [18]

Espacio de nombres	**Utilidad de los tipos de datos que contiene**
System	Tipos muy frecuentemente usados, como los tipos básicos, tablas, excepciones, fechas, números aleatorios, recolector de basura, entrada/salida en consola, etc.
System.Collections	Colecciones de datos de uso común como pilas, colas, listas, diccionarios, etc.
System.Data	Manipulación de bases de datos. Forman la denominada arquitectura **ADO.NET**.
System.IO	Manipulación de archivos y otros flujos de datos.
System.Net	Realización de comunicaciones en red.
System.Reflection	Acceso a los metadatos que acompañan a los módulos de código.
System.Runtime.Remoting	Acceso a objetos remotos.
System.Security	Acceso a la política de seguridad en que se basa el CLR.
System.Threading	Manipulación de hilos.
System.Web.UI.WebControls	Creación de interfaces de usuario basadas en ventanas para aplicaciones Web.
System.Winforms	Creación de interfaces de usuario basadas en ventanas para aplicaciones estándar.
System.XML	Acceso a datos en formato XML.

Common Type System (CTS)

El Common Type System (CTS) o Sistema de Tipos Común es el conjunto de reglas que han de seguir las definiciones de tipos de datos para que el CLR las acepte. Es decir, aunque cada lenguaje gestionado disponga de su propia sintaxis para definir tipos de datos, en el MSIL resultante de la compilación de sus códigos fuente se han de cumplir las reglas del CTS. Algunos ejemplos de estas reglas son: [18]

- Cada tipo de dato puede constar de cero o más miembros. Cada uno de estos miembros puede ser un campo, una propiedad (Get y/o Set), un método o un evento.

- No puede haber herencia múltiple, y todo tipo de dato ha de heredar directa o indirectamente de **System.Object**.
- Los modificadores de acceso son:

Modificador	Código desde el que es accesible el miembro
public	Cualquier código.
private	Código del mismo tipo de dato.
family	Código del mismo tipo de dato o de hijos de éste.
assembly	Código del mismo ensamblado.
family and assembly	Código del mismo tipo o de hijos de éste ubicado en el mismo ensamblado.
family or assembly	Código del mismo tipo o de hijos de éste, o código ubicado en el mismo ensamblado.

Common Language Specification (CLS)

El **Common Language Specification** (CLS) o Especificación del Lenguaje Común es un conjunto de reglas que han de seguir las definiciones de tipos que se hagan usando un determinado lenguaje gestionado si se desea que sean accesibles desde cualquier otro lenguaje gestionado. Obviamente, sólo es necesario seguir estas reglas en las definiciones de tipos y miembros que sean accesibles externamente, y no en la de los privados. Además, si no importa la interoperabilidad entre lenguajes tampoco es necesario seguirlas. A continuación se listan algunas de las reglas significativas del CLS:

- Los tipos de datos básicos admitidos son **bool**, **char**, **byte**, **short**, **int**, **long**, **float**, **double**, **string** y **object**. Nótese pues que no todos los lenguajes tienen porqué admitir los tipos básicos enteros sin signo o el tipo **decimal** como lo hace C#. [18]
- Las tablas han de tener una o más dimensiones, y el número de dimensiones de cada tabla ha de ser fijo. Además, han de indexarse empezando a contar desde 0. [18]
- Se pueden definir tipos abstractos y tipos sellados. Los tipos sellados no pueden tener miembros abstractos. [18]
- Las excepciones han de derivar de **System.Exception**, los delegados de **Ssytem.Delegate**, las enumeraciones de **System.**Enum, y los tipos por valor que no sean enumeraciones de **System.ValueType**. [18]
- Los métodos de acceso a propiedades en que se traduzcan las definiciones get/set de éstas han de llamarse de la forma **get_X** y **set_X** respectivamente, donde X es el nombre de la propiedad; los de acceso a indicadores han de traducirse en métodos **get_Item** y **set_Item**; y en el caso de los eventos, sus definiciones add/remove han de traducirse en métodos de **add_X** y **remove_X**. [18]

- En las definiciones de atributos sólo pueden usarse enumeraciones o datos de los siguientes tipos: **System.Type, string, char, bool, byte, short, int, long, flota, double** y **object.** [18]
- En un mismo ámbito no se pueden definir varios identificadores cuyos nombres sólo difieran en la capitalización usada. De este modo se evitan problemas al acceder a ellos usando lenguajes no sensibles a mayúsculas. [18]
- Las enumeraciones no pueden implementar interfaces, y todos sus campos han de ser estáticos y del mismo tipo. El tipo de los campos de una enumeración sólo puede ser uno de estos cuatro tipos básicos: **byte, short, int** o **long.** [18]

Proyecto Mono

Hace algunos años, la gente del mundo del código abierto se dio cuenta que la idea de .NET era buena. El hecho de poder tener una infraestructura común para el desarrollo y la ejecución de aplicaciones, independiente del sistema operativo, es realmente una ventaja. Entonces, pusieron manos a la obra. [18]

Se planteo si era bueno hacer una implementación de .NET de código fuente abierto o si era mejor desarrollar una nueva tecnología, los creadores de Mono creyeron que era bueno hacer una implementación de código fuente abierto. Las razones son varias, pero entre ellas, podemos mencionar que tanto la idea como la implementación llevadas a cabo por Microsoft son buenas, por lo cual no era necesario encarar un rediseño completo. [18]

Con el objetivo ya definido, la gente de Ximian (empresa adquirida por Novell) comenzó el desarrollo de un proyecto realmente ambicioso.

Microsoft ha decidido liberar todas las especificaciones bajo el estándar ECMA 335 y así es como aparece Mono. Y, aunque muchos no lo crean, los desarrolladores de este proyecto (una comunidad de código fuente abierto) y Microsoft (una empresa que de código abierto tiene poco y nada) trabajan juntos para que esta plataforma sea cada día mejor y más estable.

El estándar ECMA 335 define una infraestructura **de lenguaje común** (Common Language Infrastructure, CLI) en la cual aplicaciones escritas en diferentes lenguajes de alto nivel pueden ejecutarse en distintos entornos sin necesidad de rescribir el código. Entre otras cosas, en este estándar se definen un formato de archivo, librerías y un lenguaje intermedio.

Mono ofrece a los usuarios de Windows, GNU/Linux y otras plataformas UNIX (como Mac OS X) la posibilidad de desarrollar aplicaciones utilizando herramientas que sean compatibles con .NET. Esto significa una única tecnología de desarrollo común a todos los sistemas operativos. En teoría, un .exe generado en Windows podría funcionar en GNU/Linux y lo mismo sucedería al revés. [18]

La versión actual de Mono es plenamente funcional y capaz de servirnos como herramienta de desarrollo para los proyectos más avanzados. Pero, sin lugar a dudas, lo que aún está en pañales son algunas "facilidades" realmente importantes como el set de

funciones de *System.Windows.Forms*, el desarrollo de MonoBasic y el desarrollo de interfaces visuales. (www.gotmono.com)

Mono no es el único jugador en el campo de las implementaciones de código fuente abierto de la tecnología .NET. Casi paralelamente a su creación aparece el proyecto DOTGNU.

COM y .NET

COM y .NET son tecnologías complementarias de desarrollo. El .NET Common Language Runtime provee integración bidireccional y transparente con COM. Esto significa que las aplicaciones y componentes de COM y .NET pueden utilizar la funcionalidad de cada sistema. Esto protege las inversiones existentes en aplicaciones COM mientras que permite aprovechar el uso de .NET. COM y .NET pueden alcanzar resultados similares. El framework de .NET provee a los desarrolladores un número significativo de ventajas incluyendo un modelo más robusto y seguro, un manejo automático de la memoria y soporte nativo para Servicios Web. [14]

3.2. Lenguaje C#

Origen y necesidad de un nuevo lenguaje

C# es el nuevo lenguaje de propósito general diseñado por Microsoft para su plataforma .NET. Sus principales creadores son Scout Wiltamuth y Anders Hejlsberg, este último también conocido por haber sido el diseñador del lenguaje Turbo Pascal y la herramienta RAD Delphi. [18]

Aunque es posible escribir código para la plataforma .NET en muchos otros lenguajes, C# es el único que ha sido diseñado específicamente para ser utilizado en ella, por lo que programarla usando C# es mucho más sencillo e intuitivo que hacerlo con cualquiera de los otros lenguajes ya que C# carece de elementos heredados innecesarios en .NET. Por esta razón, se suele decir que C# es el **lenguaje nativo de .NET**. [18]

La sintaxis y estructuración de C# es muy similar a la de C++, ya que la intención de Microsoft con C# es facilitar la migración de códigos escritos en estos lenguajes a C# y facilitar su aprendizaje a los desarrolladores habituados a ellos. Sin embargo, su sencillez y el alto nivel de productividad son equiparables a los de Visual Basic. [18]

Un lenguaje que hubiese sido ideal utilizar es Java, pero debido a problemas con la empresa creadora del mismo (Sun), Microsoft ha tenido que desarrollar un nuevo lenguaje que añadiese a las ya probadas virtudes de Java las modificaciones que Microsoft tenía pensado añadirle para mejorarlo aún más y hacerlo un lenguaje orientado al desarrollo de componentes. [18]

En resumen, C# es un lenguaje de programación que toma las mejores características de lenguajes preexistentes como Visual Basic, Java o C++ y las combina en uno solo. El hecho de ser relativamente reciente no implica que sea inmaduro, pues Microsoft ha escrito la mayor parte de la BCL usándolo. [18]

Características de C#

A continuación se enumeran de manera resumida las principales características de C#. Algunas de las características aquí señaladas no son exactamente propias del lenguaje sino de la plataforma .NET en general. Sin embargo, también se comentan aquí también en tanto que tienen repercusión directa en el lenguaje, aunque se indicará explícitamente cuáles son este tipo de características cada vez que se aborden: [18]

- **Sencillez:** C# elimina muchos elementos que otros lenguajes incluyen y que son innecesarios en .NET. Por ejemplo:
  - El código escrito en C# es **autocontenido**, lo que significa que no necesita de archivos adicionales al propio fuente tales como archivos de cabecera o archivos IDL.
  - El tamaño de los tipos de datos básicos es fijo e independiente del compilador, sistema operativo o máquina para quienes se compile (no como en C++), lo que facilita la portabilidad del código.
  - Se eliminaron características como macros, herencia múltiple o la necesidad de un operador diferente del (.) para acceder a miembros de espacios de nombres (::) propias de C++.
- **Modernidad:** C# incorpora en el propio lenguaje elementos que a lo largo de los años ha ido demostrándose son muy útiles para el desarrollo de aplicaciones y que en otros lenguajes como Java o C++ hay que simular, como un tipo básico **decimal** que permita realizar operaciones de alta precisión con reales de 128 bits (muy útil en el mundo financiero), la inclusión de una instrucción **foreach** que permita recorrer colecciones con facilidad y es ampliable a tipos definidos por el usuario, la inclusión de un tipo básico **string** para representar cadenas o la distinción de un tipo **bool** específico para representar valores lógicos.
- **Orientación a objetos:** como todo lenguaje de programación de propósito general actual, C# es un lenguaje orientado a objetos, aunque eso es más bien una característica del CTS que de C#. Una diferencia de este enfoque orientado a objetos respecto al de otros lenguajes como C++ es que el de C# es más puro en tanto que no admite ni funciones ni variables globales sino que todo el código y datos han de definirse dentro de definiciones de tipos de datos, lo que reduce problemas por conflictos de nombres y facilita la legibilidad del código.

  C# soporta todas las características propias del paradigma de programación orientada a objetos: **encapsulamiento, herencia** y **polimorfismo.**

  En lo referente a la encapsulación es importante señalar que aparte de los típicos modificadores **public, private** y **protected** de Java, C# añade un cuarto modificador llamado **internal**, que puede combinarse con **protected** e indica

que al elemento a cuya definición procede sólo puede accederse desde su mismo ensamblado.

Respecto a la herencia (a diferencia de C++ y al igual que Java) C# sólo admite herencia simple de clases ya que la herencia múltiple provoca más dolores de cabeza que facilidades y en la mayoría de los casos su utilidad puede ser simulada con facilidad mediante herencia múltiple de interfaces. De todos modos, esto vuelve a ser más bien una característica propia del CTS que de C#.

Por otro lado y a diferencia de Java, en C# se ha optado por hacer que todos los métodos sean por defecto no sobre escribibles (*sealed*) y que los redefinibles hayan de marcarse con el modificador **virtual** (como en C++), lo que permite evitar errores derivados de redefiniciones accidentales. Además, un efecto secundario de esto es que las llamadas a los métodos serán más eficientes por defecto al no tenerse que buscar en la tabla de funciones virtuales la implementación de los mismos a la que se ha de llamar. Otro efecto secundario es que permite que las llamadas a los métodos virtuales se puedan hacer más eficientemente al contribuir a que el tamaño de dicha tabla se reduzca.

- **Orientación a componentes:** la propia sintaxis de C# incluye elementos propios del diseño de componentes que otros lenguajes tienen que simular mediante construcciones más o menos complejas. Es decir, la sintaxis de C# permite definir cómodamente **propiedades** (similares a campos de acceso controlado), **eventos** (asociación controlada de funciones de respuesta a notificaciones) o **atributos** (información sobre un tipo o sus miembros).

- **Gestión automática de memoria:** como ya se comentó, todo lenguaje de .NET tiene a su disposición el recolector de basura del CLR. Esto tiene el efecto en el lenguaje de que no es necesario incluir instrucciones de destrucción de objetos. Sin embargo, dado que la destrucción de los objetos a través del recolector de basura es no determinística y sólo se realiza cuando éste se activa (ya sea por falta de memoria, finalización de la aplicación o solicitud explícita en el fuente), C# también proporciona un mecanismo de liberación de recursos determinista a través de la instrucción **using**.

- **Seguridad de tipos:** C# incluye mecanismos que permiten asegurar que los accesos a tipos de datos siempre se realicen correctamente, lo que evita que se produzcan errores difíciles de detectar por acceso a memoria no perteneciente a ningún objeto y es especialmente necesario en un entorno gestionado por un recolector de basura. Para ello se toman medidas del tipo:

  - Sólo se admiten **conversiones entre tipos compatibles**. Esto es, entre un tipo y antecesores suyos, entre tipos para los que explícitamente se haya definido un operador de conversión, y entre un tipo y un tipo hijo suyo del que un objeto del primero almacenase una referencia del segundo (**downcasting**). Obviamente, lo último sólo puede comprobarlo en tiempo de ejecución el CLR y no el compilador, por lo que en realidad el CLR y el compilador colaboran para asegurar la corrección de las conversiones.

  - No se pueden usar **variables no inicializadas**. El compilador da a los campos un valor por defecto consistente y controla mediante análisis del flujo de control del fuente que no se acceda a una variable local sin que se le haya asignado previamente algún valor.

- Se comprueba que todo **acceso a los elementos de una tabla** se realice con índices que se encuentren dentro del rango de la misma.
- Se puede controlar la **producción de desbordamientos** (*overflow*) en operaciones aritméticas, informándose de ellos con una excepción cuando ocurra. Sin embargo, para conseguirse un mayor rendimiento en la aritmética estas comprobaciones no se hacen por defecto al operar con variables sino sólo con constantes (se pueden detectar en tiempo de compilación).
- A diferencia de Java, C# incluye **delegados**, que son similares a los punteros a funciones de C++ pero siguen un enfoque orientado a objetos, pueden almacenar referencias a varios métodos simultáneamente, y se comprueba que los métodos a los que apunten tengan parámetros y valor de retorno del tipo indicado al definirlos.
- Pueden definirse métodos que admitan un número indefinido de parámetros de un cierto tipo, y a diferencia de lenguajes como C/C++, en C# siempre se comprueba que los valores que se les pasen en cada llamada sean de los tipos apropiados.

- **Instrucciones seguras:** para evitar errores muy comunes, en C# se han impuesto una serie de restricciones en el uso de las instrucciones de control más comunes. Por ejemplo, la guarda de toda condición ha de ser una expresión condicional y no aritmética, con lo que se evitan errores por confusión del operador de igualdad (==) con el de asignación (=); y todo caso de un **switch** ha de terminar en un **break** o **goto** que indique cuál es la siguiente acción a realizar, lo que evita la ejecución accidental de casos y facilita su reordenación.

- **Sistema de tipos unificado:** a diferencia de C++, en C# todos los tipos de datos que se definan siempre derivarán, aunque sea de manera implícita, de una clase base común llamada **System.Object**, por lo que dispondrán de todos los miembros definidos en ésta clase (es decir, serán "objetos").

  En C# esto también es aplicable a los tipos de datos básicos. Además, para conseguir que ello no tenga una repercusión negativa en su nivel de rendimiento, se ha incluido un mecanismo transparente de **boxing** y **unboxing** con el que se consigue que sólo sean tratados como objetos cuando la situación lo requiera, y mientras tanto puede aplicárseles optimizaciones específicas.

  El hecho de que todos los tipos del lenguaje deriven de una clase común facilita enormemente el diseño de colecciones genéricas que puedan almacenar objetos de cualquier tipo.

- **Extensibilidad de tipos básicos:** C# permite definir, a través de **estructuras**, tipos de datos para los que se apliquen las mismas optimizaciones que para los tipos de datos básicos. Es decir, que se puedan almacenar directamente en la pila (luego su creación, destrucción y acceso serán más rápido) y se asignen por valor y no por referencia. Para conseguir que lo último no tenga efectos negativos al pasar estructuras como parámetros de métodos, se da la posibilidad de pasar referencias a pila a través del modificador de parámetro **ref**.

- **Extensibilidad de operadores:** para facilitar la legibilidad del código y conseguir que los nuevos tipos de datos básicos que se definan a través de las estructuras estén al mismo nivel que los básicos predefinidos en el lenguaje, al igual que en C++ y a diferencia de Java, C# permite redefinir el significado de la

mayoría de los operadores (incluidos los de conversión, tanto para conversiones implícitas como explícitas) cuando se apliquen a diferentes tipos de objetos.

Las redefiniciones de operadores se hacen de manera inteligente, de modo que a partir de una única definición de los operadores ++ y – el compilador puede deducir automáticamente como ejecutarlos de manera prefijas y postfija; y definiendo operadores simples (como +), el compilador deduce cómo aplicar su versión de asignación compuesta (+=). Además, para asegurar la consistencia, el compilador vigila que los operadores con opuesto siempre se redefinan por parejas (por ejemplo, si se redefine ==, también hay que redefinir !=).

También se da la posibilidad, a través del concepto de **indicador**, de redefinir el significado del operador [] para los tipos de datos definidos por el usuario, con lo que se consigue que se pueda acceder al mismo como si fuese una tabla. Esto es muy útil para trabajar con tipos que actúen como colecciones de objetos.

- **Extensibilidad de modificadores:** C# ofrece, a través del concepto de **atributos**, la posibilidad de añadir a los metadatos del módulo resultante de la compilación de cualquier fuente, información adicional a la generada por el compilador que luego podrá ser consultada en tiempo de ejecución a través de la librería de reflection de .NET. Esto, que más bien es una característica propia de la plataforma .NET y no de C#, puede usarse como un mecanismo para definir nuevos modificadores.

- **Versionado:** C# incluye una **política de versionado** que permite crear nuevas versiones de tipos sin temor a que la introducción de nuevos miembros provoquen errores difíciles de detectar en tipos hijos previamente desarrollados y ya extendidos con miembros de igual nombre a los recién introducidos.

  Si una clase introduce un nuevo método cuyas redefiniciones deban seguir la regla de llamar a la versión de su padre en algún punto de su código, difícilmente seguirán esta regla miembros de su misma firma definidos en clases hijas previas a la definición del mismo en la clase padre; o si introduce un nuevo campo con el mismo nombre que algún método de una clase hija, la clase hija dejará de funcionar. Para evitar que esto ocurra, en C# se toman dos medidas:

  - Se obliga a que toda redefinición deba incluir el modificador **override**, con lo que la versión de la clase hija nunca sería considerada como una redefinición de la versión de miembro en la clase padre ya que no incluiría **override**. Para evitar que por accidente un programador incluya este modificador, sólo se permite incluirlo en miembros que tengan la misma firma que miembros marcados como redefinibles mediante el modificador **virtal**. Así además se evita el error tan frecuente en Java de creerse haber redefinido un miembro, pues si el miembro con **override** no existe en la clase padre se producirá un error de compilación.

  - Si no se considera redefinición, entonces se considera que lo que se desea es ocultar el método de la clase padre, de modo que para la clase hija sea como si nunca hubiese existido. El compilador avisará de esta decisión a través de un mensaje de aviso que puede suprimirse incluyendo el modificador **new** en la definición del miembro en la clase hija para así indicarle explícitamente la intención de ocultación.

- **Eficiente:** en principio, en C# todo el código incluye numerosas restricciones para asegurar su seguridad y no permite el uso de punteros. Sin embargo, y a

diferencia de Java, en C# es posible saltarse dichas restricciones manipulando objetos a través de punteros. Para ello basta marcar regiones de código como inseguras (modificador **unsafe**) y podrán usarse en ellas punteros de forma similar a cómo se hace en C++, lo que puede resultar vital para situaciones donde se necesite una eficiencia y velocidad de procesamiento muy grandes.

- **Compatible:** para facilitar la migración de programadores, C# no sólo mantiene una sintaxis muy similar a C, C++ o Java que permite incluir directamente en código escrito en C# fragmentos de código escrito en estos lenguajes, sino que el CLR también ofrece, a través de los llamados **Platform Invocation Services (PInvoke)**, la posibilidad de acceder a código nativo escrito como funciones sueltas no orientadas a objetos tales como las DLLs de la API Win32. Nótese que la capacidad de usar punteros en código inseguro permite que se pueda acceder con facilidad a este tipo de funciones, ya que éstas muchas veces esperan recibir o devuelven punteros.

  También es posible acceder desde código escrito en C# a objetos COM. Para facilitar esto, el *.NET Framework SDK* incluye unas herramientas llamadas **tlbimp** y **regasm** mediante las que es posible generar automáticamente clases Proxy que permitan, respectivamente, usar objetos COM desde .NET como si de objetos .NET se tratase y registrar objetos .NET para su uso desde COM.

  Finalmente, también se da la posibilidad de usar controles ActiveX desde código .NET y viceversa. Para lo primero se utiliza la utilidad **aximp**, mientras que para lo segundo se usa la ya mencionada **regasm**.

3.3. ASP.NET

La evolución de ASP

Active Server Pages (ASP) es una tecnología relativamente actual introducida en 1996. Antes de ASP, los desarrolladores eran capaces de crear sitios Web dinámicos sobre plataformas Microsoft usando **Common Gateway Interface** (CGI) e **Internet Server Application Programming Interface** (ISAPI), cada uno de los cuales formó parte de la evolución de ASP.

CGI fue el primer método aceptado ampliamente para entregar contenido Web dinámico. Es efectivamente un método para extender la funcionalidad de un servidor Web habilitándolo a generar respuestas HTTP dinámicas usando un programa típicamente escrito en C o un lenguaje de scripting. Esto permitió que el contenido de una página sea personalizado por el usuario y construido desde información almacenada en una base de datos. Aunque es poderoso, CGI tiene varios defectos. Para cada solicitud HTTP que recibe una aplicación CGI se crea un nuevo proceso. Después que la solicitud fue atendida, el proceso es destruido. Crear y destruir procesos repetidamente ha demostrado ser una carga tremenda aún para los servidores Web más calificados.

La plataforma Active Server de Microsoft fue apuntada a resolver las limitaciones técnicas de la programación CGI. Esta plataforma incluía un conjunto de herramientas que los desarrolladores podían utilizar para escribir aplicaciones Web. Sin embargo, esta plataforma no incluyó originalmente ASP. Los desarrolladores fueron forzados a escribir extensiones o filtros ISAPI.

Las extensiones ISAPI y CGI son muy similares, salvó principalmente por una excepción. Las aplicaciones CGI eran generalmente implementadas como ejecutables (EXEs) sobre la plataforma Windows, mientras que las extensiones ISAPI son implementadas como librerías de enlaces dinámicas (DLLs), lo que significa que son cargadas en memoria solo una vez, por demanda, y permanecen residentes en el mismo proceso que el Internet Information Server (IIS). Por lo tanto, las extensiones ISAPI no sufren los mismos problemas de performance que las aplicaciones CGI. Además, las extensiones ISAPI son multihilo, lo que significa que pueden manejar solicitudes concurrentes sin degradar la performance del sistema.

Al igual que las extensiones ISAPI, los filtros ISAPI son multihilo, implementados como DLLs y corren en el mismo espacio de memoria que el IIS. Sin embargo, los filtros ISAPI no son invocados por solicitudes de los clientes. En lugar de eso, filtran o interpretan y opcionalmente procesan solicitudes HTTP. Sin embargo, como los filtros ISAPI actúan sobre todas las solicitudes HTTP, deben ser usados con moderación para evitar severos problemas de performance.

Así como las extensiones y los filtros ISAPI son útiles y potentes, pueden ser difíciles de desarrollar por programadores principiantes. Reconociendo esto, Microsoft liberó varios productos de desarrollo para la plataforma basados sobre ISAPI. Estos tuvieron una vida corta e incluyeron dbWeb e Internet Database Connector (IDC) que evolucionaron hacia ASP. En 1996 Microsoft lanzó ASP. ASP permitió a los desarrolladores ejecutar código dentro de una página Web. La tecnología ASP

evolucionó hasta convertirse en uno de los productos para desarrollar sitios Web dinámicos. Esto es probablemente debido a la facilidad con que páginas y aplicaciones complejas pueden ser creadas combinando la posibilidad de usar componentes a través de la arquitectura COM/COM+.

Desde 1996 ha habido varias versiones de ASP. En 1998 Microsoft introdujo ASP 2.0 como parte de Windows NT 4.0 Option Pack. Con ASP 2.0 y IIS 4.0, las aplicaciones ASP y sus componentes asociados pudieron ser creados en un espacio de memoria separado del espacio de memoria del servidor Web para mejorar la tolerancia a fallas. En el 2000, Microsoft reveló 3.0. Las diferencias entre las capacidades de ASP 2.0 y 3.0 parecen ser muy limitadas. Sin embargo, correr sobre Windows 2000 mejoró en gran medida la performance.

Así como, ASP es poderoso y increíblemente simple de usar, tiene los siguientes inconvenientes:

- **El código ASP puede complicarse muy rápidamente.** El código ASP tiende a desestructurarse y desordenarse. Código del lado del servidor se entremezcla con script del lado del cliente y HTML.
- **Para hacer cualquier cosa en ASP hay que escribir código.** ASP no tiene un modelo de componentes.
- **El código está mezclado con la presentación.** Esto causa problemas cuando los desarrolladores y los diseñadores trabajan juntos. Además, soportar internacionalización y múltiples tipos de clientes es difícil.
- **La combinación ASP y IIS no siempre es la plataforma más confiable.** Desarrollar una aplicación ASP es una cosa, desarrollar una aplicación ASP buena, eficiente y confiable es otra. De cualquier manera, la tolerancia a fallas de ASP es un poco mejor.
- **Instalar una aplicación ASP que utilice COM puede ser dificultoso.** Los objetos COM deben registrarse y son bloqueados por el sistema operativo cuando están siendo usados. Como resultado de esto, manejar una aplicación productiva, especialmente en un Web Farm, o aplicación Web que utilice más de un servidor, puede ser muy dificultoso.

Los beneficios de ASP.NET

Microsoft, dándose cuenta que ASP poseía algunos defectos significantes, desarrollo ASP.NET. ASP.NET es un conjunto de componentes que provee a los desarrolladores de un framework con el cual pueden implementar funcionalidades complejas.

ASP.NET es una plataforma de desarrollo Web unificada que provee los servicios necesarios a los desarrolladores para construir aplicaciones Web. Mientras es en gran parte sintácticamente compatible con ASP, también provee un nuevo modelo de programación e infraestructura para aplicaciones más seguras, escalables y estables [21].

ASP.NET también forma parte de la plataforma .NET y está totalmente integrado con la misma.

Dos de las principales mejoras de ASP.NET sobre el ASP tradicional son la escalabilidad y la disponibilidad. ASP.NET es escalable en cuanto a que provee servicios de estado que pueden ser utilizados para manejar variables de sesión a través de múltiples servidores Web. Además, ASP.NET provee un modelo de proceso de alta performance que puede detectar fallas de aplicación y recuperarse de ellas [4].

Junto con las mejoras de escalabilidad y disponibilidad, ASP.NET provee los siguientes beneficios adicionales [4]:

- **Desarrollo simplificado.** ASP.NET ofrece un modelo de objetos muy rico que los desarrolladores pueden usar para reducir la cantidad de código que necesitan escribir.
- **Independencia del lenguaje.** Las páginas ASP deben ser escritas con scripting. En otras palabras, las páginas ASP deben ser escritas en un lenguaje que es interpretado más que compilado. ASP.NET permite usar lenguajes compilados obteniendo una mejor performance y compatibilidad entre lenguajes.
- **Instalación simplificada.** Con componentes .NET, la instalación es tan fácil como copiar el ensamblado del componente en la ubicación deseada.
- **Compatibilidad entre clientes.** Uno de los principales problemas que enfrentan los desarrolladores hoy en día es escribir código que pueda ser dibujado correctamente sobre varios tipos de clientes. Por ejemplo, escribir un script que se dibuje correctamente en Internet Explorer 5.5 y Netscape Navigator 4.7, sobre una PDA y un teléfono móvil es muy difícil, sino imposible además de consumir mucho tiempo. ASP.NET provee una abundante cantidad de componentes de servidor que pueden producir automáticamente una salida apuntada específicamente a cada tipo de cliente.
- **Servicios Web.** ASP.NET provee características que permiten a los desarrolladores crear servicios Web que puedan ser consumidos por cualquier cliente que entienda HTTP y XML.
- **Performance.** Las páginas ASP.NET son compiladas mientras que las páginas ASP son interpretadas. Cuando una página ASP.NET es solicitada por primera vez, esta es compilada y puesta en caché por el CLR. Esta copia puesta en caché puede entonces ser reusada por cada solicitud siguiente de la página. La performance es mejorada porque después de la primera solicitud, el código puede correr mucho más rápido desde su versión compilada.

4. Servicios Web en .Net

4.1. Introducción

El mundo del desarrollo de aplicaciones en entorno Internet ha pasado por distintas etapas desde los albores del protocolo HTTP, los navegadores y los documentos HTML. En la última de ellas, que es la que vivimos actualmente, el panorama del desarrollo de aplicaciones Web se caracteriza por la gran sofisticación que han alcanzado los servidores y las herramientas de desarrollo asociadas. Por un lado, desde el punto de vista del diseño, en el que contamos con herramientas visuales de todo tipo y editores HTML que harían las delicias de los viejos programadores. Por otro, desde el punto de vista del desarrollo, con lenguajes, librerías y generadores de código de todo tipo, y con servidores web y de aplicaciones con infinidad de ayudas para el acceso a sistemas de base de datos, "poolings" de objetos, caché de páginas y datos, soporte transaccional, etc.

Esta etapa ha dado múltiples aplicaciones y portales de gran calidad y rendimiento, y son muchos los desarrolladores que han dejado lo mejor de sí en estos proyectos. Son muchas las exigencias de los usuarios y clientes en relación a sus proyectos y desarrollos, y hoy en día, un portal vertical "medio" debe trabajar con una serie de parámetros de velocidad, rendimiento, seguridad, navegabilidad y disponibilidad que hacen de los sistemas avanzados de desarrollo un elemento fundamental a la hora de conseguir la realización de las metas tecnológicas fijadas.

Pero si algo define a esta etapa desde un punto de vista negativo, es el aislamiento que vive cada uno de estos desarrollos. Cada vez que pensamos en realizar un portal vertical, pensamos en el público objetivo, en sus gustos y preferencias, en un diseño claro y eficaz, en una navegabilidad y en una usabilidad. También trabajamos en un modelo de datos eficaz, en una capa de reglas de negocio extensible, y en un rendimiento global aceptable. Pero pocas veces pensamos en cómo va a comunicarse nuestro portal con el resto de Internet, y no hablamos de los clientes de Internet, los que entrarán en nuestro portal particular, sino que hablamos del "resto" de servidores de Internet [20].

Con los **Servicios Web**, los orígenes de información se convierten en componentes que podemos usar, reusar, mezclar y comparar para enriquecer las aplicaciones Internet e intranet.

Cada proveedor u organización de estándares define a los Servicios Web de una forma un tanto diferente. Por ejemplo, Gartner define a los Servicios Web como "*componentes de software levemente acoplados que interactúan unos con otros dinámicamente por medio de tecnologías de Internet estándar*". Forrester Research toma una aproximación más abierta para definir a los Servicios Web como "*conexiones automatizadas entre personas, sistemas y aplicaciones que exponen funcionalmente elementos de negocio como servicios de software y crean nuevos valores de negocio*" [43].

Un buen lugar para comenzar una exploración de Servicios Web es definir precisamente qué es un Servicio Web. Un Servicio Web es una aplicación que:

- corre sobre un servidor Web.
- expone métodos Web.
- escucha requerimientos HTTP que invocan métodos Web.
- ejecuta métodos Web y retorna el resultado.

4.2. Servicios Web

Los clientes se comunican con el servidor sobre Internet afrontando nuevas barreras para comunicarse con él. Los administradores de redes alrededor del mundo, concientes de la seguridad, han introducido routers y firewalls corporativos impidiendo prácticamente todo tipo de comunicación sobre Internet. Estos administradores a menudo se rehusan a abrir puertos más allá de lo mínimo necesario.

Si no tenemos la suerte suficiente para que el administrador de red abra los puertos apropiados, protocolos propietarios tales como los usados por DCOM, CORBA y Java RMI no son prácticos para escenarios de Internet.

Otros problemas con estas tecnologías es que son inherentemente orientadas a la conexión y por lo tanto no pueden manejar exitosamente interrupciones en la red. Debido a que Internet no esta bajo nuestro control directo, no podemos hacer ninguna asunción sobre la calidad y confiabilidad de la conexión. Si ocurre una interrupción de red, la siguiente llamada del cliente al servidor puede ser que falle.

La naturaleza orientada a la conexión de estas tecnologías también hace desafiante construir una infraestructura con el balanceo de carga necesario para alcanzar una alta escalabilidad. Una vez que la conexión entre el cliente y el servidor está hecha, no podemos simplemente dirigir el siguiente requerimiento hacia otro servidor.

Los desarrolladores han tratado de superar estas limitaciones con un modelo llamado **stateless programming** (programación sin estados), pero han tenido un éxito limitado porque las tecnologías son bastante pesadas y se hace muy costoso reestablecer la conexión con un objeto remoto.

En un esfuerzo para mejorar el soporte a tales escenarios, Microsoft adoptó inicialmente la estrategia de aumentar sus tecnologías existentes incluyendo COM Internet Services (CIS), que nos permite establecer una conexión DCOM entre el cliente y el componente remoto sobre el puerto 80. Por varias razones, CIS no fue ampliamente aceptado [6].

Estaba claro que una nueva aproximación sería necesaria. Así que Microsoft se decidió a dirigir el problema hacia [6]:

- *Interoperabilidad*: el servicio remoto debe ser capaz de ser consumido por clientes sobre otras plataformas.
- *Amenizar con Internet*: la solución debe trabajar bien soportando clientes que accedan al servicio remoto desde Internet.
- *Interfaces fuertemente tipadas*: no deben ser ambiguos los tipos de datos enviados hacia y recibidos desde el servicio remoto. Además, los tipos de datos definidos por el servicio remoto se deben enlazar razonablemente bien con los

tipos de datos definidos por la mayoría de los lenguajes de programación procedural.

- *Utilizar los estándares de Internet existentes*: la implementación del servicio remoto debe utilizar los estándares de Internet existentes tanto como sea posible y debe evitar reinventar soluciones a problemas que ya fueron resueltos. Una solución construida sobre estándares de Internet ampliamente aceptados puede utilizar herramientas existentes y productos creados para la tecnología.
- *Soporte a cualquier lenguaje*: la solución no debe estar fuertemente acoplada a un lenguaje de programación particular. Java RMI, por ejemplo, está fuertemente acoplado al leguaje Java, lo que dificulta invocar funcionalidad sobre un objeto remoto Java desde Visual Basic o Perl. Un cliente debe ser capaz de implementar un nuevo Servicio Web o usar un Servicio Web existente sin importar el lenguaje de programación en el que el cliente fue escrito.
- *Soporte a cualquier infraestructura de componentes distribuidos*: la solución no debe estar acoplada a una infraestructura de componentes en particular. Los protocolos subyacentes deben facilitar un nivel de comunicación base entre las infraestructuras de objetos distribuidos existentes tales como DCOM y CORBA.

Los Servicios Web son el punto principal de la iniciativa Microsoft .NET. Ellos son la clave en la visión del mundo, para Microsoft, en el que las computadoras dialogan unas con otras a través de la Web usando HTTP y otros protocolos soportados universalmente.

Un Servicio Web es un tipo diferente de aplicación Web. No tiene una interfaz de usuario tradicional y expone funciones invocables a través de la Web conocidas como **métodos Web**. Además, los Servicios Web no están diseñados para servir a usuarios finales como hacen las aplicaciones Web tradicionales, sino para proveer servicios a otras aplicaciones como son las aplicaciones Web, aplicaciones GUI o inclusive aplicaciones de línea de comandos. Los tipos de servicios que pueden proveer los Servicios Web son muy variados, pueden desde validar tarjetas de crédito hasta proveer información sobre el clima. Microsoft, Sun, IBM y otros prevén un mundo en el que toda la información este disponible vía Servicios Web.

Una aplicación que hable el lenguaje de los Servicios Web tiene acceso a un universo de servicios que está emergiendo. Hoy en día ya hay compañías alrededor del mundo que exponen contenido y lógica de negocio a través de Servicios Web.

Los Servicios Web tienen el potencial para cambiar el mundo como unas pocas tecnologías lo han hecho.

4.3. Bloques de construcción de los Servicios Web

El gráfico siguiente muestra los bloques de construcción base necesarios para facilitar la comunicación remota entre dos aplicaciones:

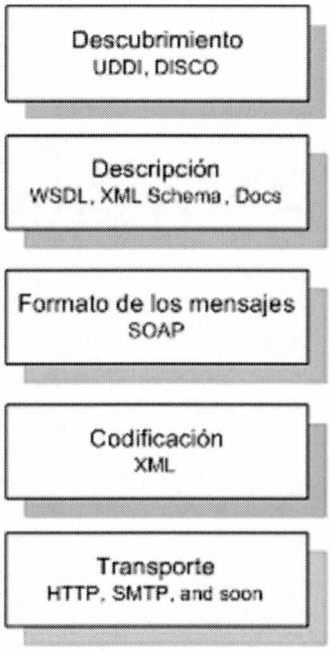


Figura 19. Bloques de construcción.

Veamos el propósito de cada uno de estos bloques mencionando el equivalente DCOM de cada uno de ellos:

- *Descubrimiento*: la aplicación cliente que necesite acceder a la funcionalidad expuesta por un Servicio Web necesita una forma de resolver la ubicación del Servicio Web. Esto es llevado a cabo a través de un proceso generalmente llamado **descubrimiento** (discovery). El descubrimiento puede ser facilitado vía un directorio centralizado así como por métodos para tal fin. En DCOM, el Service Control Manager (SCM) provee este servicio. [6]

- *Descripción*: una vez que ha sido resuelto el punto anterior para un Servicio Web en particular, el cliente necesita la información suficiente para poder interactuar con él apropiadamente. La descripción del Servicio Web comprende metadata estructurada sobre la interfase que se pretende que sea consumida por una aplicación cliente así como la documentación escrita sobre el Servicio Web incluyendo ejemplos de uso. Los componentes DCOM exponen metadata estructurada sobre sus interfases vía una *librería de tipos* (typelib). La metadata dentro de la typelib de un componente es almacenada en un formato binario propietario y es accedida vía una API propietaria. [6]

- *Formato de los mensajes*: para intercambiar datos, el cliente y el servidor tienen que estar de acuerdo sobre una forma común de codificar y dar formato a los mensajes. Una forma estándar de codificar datos asegura que los datos codificados por el cliente serán interpretados apropiadamente por el servidor. En DCOM, los mensajes enviados entre un cliente y un servidor tienen el formato definido por el protocolo DCOM Object RPC (ORPC). Sin una forma estándar de dar formato a los mensajes, desarrollar herramientas para abstraer al desarrollador de los protocolos subyacentes es imposible. Crear una capa de abstracción entre el desarrollador y los protocolos subyacentes permite al desarrollador enfocarse más en los problemas del negocio que en la infraestructura requerida para implementar la solución. [6]

- *Codificación*: los datos transmitidos entre el cliente y el servidor necesitan ser codificados dentro del cuerpo del mensaje. DCOM usa un esquema de codificación binaria para serializar los datos contenidos por los parámetros intercambiados entre el cliente y el servidor. [6]
- *Transporte*: una vez que el mensaje tiene formato y los datos han sido serializados dentro del cuerpo del mensaje, este debe ser transportado entre el cliente y el servidor sobre algún protocolo de transporte. DCOM soporta varios protocolos de propietarios enlazados a varios protocolos de red tales como TCP, SPX, NetBEUI y NetBIOS sobre IPX. [6]

4.4. Decisiones de diseño

Eligiendo el protocolo de transporte

El primer paso fue determinar como se comunicarían el cliente y el servidor. El cliente y el servidor pueden residir sobre la misma LAN, pero el cliente puede comunicarse con el servidor sobre Internet. Por lo tanto, el protocolo de transporte debe ser apropiado para ambientes LAN e Internet.

Tecnologías como DCOM. CORBA y Java RMI no son apropiadas para soportar la comunicación entre el cliente y el servidor sobre Internet. Protocolos tales como Hypertext Transfer Protocol (HTTP) y Simple Mail Transfer Protocol (SMTP) son protocolos de Internet probados. HTTP define un patrón de mensajes requerimiento-respuesta para presentar un requerimiento y dar una respuesta. SMTP define un protocolo de mensajes direccionables para comunicación asincrónica. SMTP es el protocolo utilizado para correo electrónico.

Las aplicaciones Web basadas en HTTP son intrínsecamente sin estado. Estas no se basan sobre una conexión continua entre el cliente y el servidor, esto hace a HTTP un protocolo ideal para alta confiabilidad con configuraciones tales como firewalls. Si el servidor que manejó originalmente el requerimiento del cliente se pone no disponible, los requerimientos siguientes pueden ser automáticamente direccionados a otro servidor sin que el cliente se de cuenta. [6]

Casi todas las compañías tienen una infraestructura que soporta SMTP. SMTP es apropiado para comunicación asincrónica. Si el servicio es desestabilizado, la infraestructura de correo electrónico maneja automáticamente los reintentos. A diferencia con HTTP, podemos pasar mensajes SMTP a un servidor de correo local que intentará entregar el mensaje de correo en nuestra representación. [6]

La otra ventaja significativa de ambos protocolos es su penetrabilidad. Los empleados están familiarizados con el correo electrónico y con los navegadores Web, y los administradores de red están cómodos soportando estos servicios. Tecnologías tales como Network Address Translation (NAT) (Traducción de direcciones de red) y los servidores proxy proveen una forma de acceso a Internet vía HTTP desde dentro de redes corporativas aisladas. Los administradores a menudo exponen un servidor SMTP

que reside dentro del firewall. Los mensajes enviados a este servidor serán direccionados a sus destinos finales vía Internet. [6]

Eligiendo el esquema de codificación

HTTP y SMTP proveen una manera de enviar datos entre el cliente y el servidor. Sin embargo, ninguno de los dos especifica como los datos dentro del cuerpo del mensaje deben ser codificados. Microsoft necesitó un estándar independiente de la plataforma para codificar los datos intercambiados entre el cliente y el servidor. [6]

Como la meta fue darle empuje a protocolos basados en Internet, **Extensible Markup Language** (XML) fue la elección natural. XML ofrece muchas ventajas, incluyendo soporte a través de la plataforma, un sistema común de tipos y soporte a un conjunto de caracteres estándar de la industria. [6]

Esquemas de codificación binaria tales como los usados por DCOM, CORBA y Java RMI deben manejar temas de compatibilidad entre diferentes plataformas de hardware. Por ejemplo, diferentes plataformas de hardware tienen diferente representación binaria interna de números de múltiples bytes. [6]

XML evita estos problemas porque usa un esquema de codificación basado en texto. También, algunos protocolos de transporte, tales como SMTP, pueden contener solo mensajes de texto. [6]

Los métodos binarios de codificación, tales como los usados por DCOM y CORBA, son engorrosos y requieren una infraestructura para abstraer a los desarrolladores de los detalles. XML es mucho más ligero y fácil de manejar porque puede ser creado y consumido usando técnicas estándar de análisis de texto.

Además, una gran variedad de analizadores de XML están disponibles sobre prácticamente todas las plataformas modernas.

Eligiendo la convención del formato

A menudo es necesario agregar metadata adicional en el cuerpo del mensaje. Por ejemplo, podemos querer incluir información sobre los tipos de servicios que el Servicio Web necesita para cumplir con el propósito de los requerimientos, tales como alistarse en una transacción o información de direccionamiento. XML no provee mecanismos para diferenciar el cuerpo del mensaje de sus datos asociados.

Protocolos de transporte como HTTP proveen un mecanismo extensible para datos de encabezado, pero algunos datos asociados con el mensaje no pueden ser especificados en el protocolo de transporte. Por ejemplo, el cliente puede enviar un mensaje que necesita ser direccionado a múltiples destinos, potencialmente sobre diferentes protocolos de transporte. Si la información de direccionamiento fue ubicada dentro del encabezado de HTTP, tendría que ser traducida antes de enviarse al siguiente intermediario sobre otro protocolo de transporte. Debido a que la información de

direccionamiento es específica para el mensaje y no al protocolo de transporte, esta debe ser parte del mensaje. [6]

Simple Object Access Protocol (SOAP) provee una manera de asociar la información del encabezado con el cuerpo del mensaje. Todo mensaje SOAP debe definir una envoltura (envelope). La envoltura tiene un cuerpo que contiene la carga útil del mensaje y un encabezado que puede contener metadata asociada con el mensaje. [6]

SOAP no impone restricciones sobre como debe ser el formato del cuerpo del mensaje. Esto es una consideración potencial porque sin una forma consistente de codificar los datos es difícil desarrollar un conjunto de herramientas que nos abstraigan de los protocolos subyacentes. [6]

Fue necesaria una forma estándar para darle formato a los mensajes en las llamadas a procedimientos remotos (RPC) que veremos más adelante.

Eligiendo un mecanismo de descripción

SOAP provee una forma estándar de dar formato a los mensajes intercambiados entre un Servicio Web y un cliente. Sin embargo, el cliente necesita información adicional para poder serializar apropiadamente el requerimiento e interpretar la respuesta. El XSD (Extended Service Desciption) provee un método para crear esquemas que pueden ser usados para describir el contenido de un mensaje. [6]

El XSD provee un conjunto de tipos de datos integrados que pueden ser usados para describir el contenido de un mensaje, además podemos crear nuestros propios tipos.

Un esquema contiene un conjunto de tipos de datos y definiciones de elementos. Un Servicio Web usa el esquema no solo para comunicar el tipo de datos que es esperado dentro del mensaje sino que también valida los mensajes entrantes y salientes.

Sin embargo un esquema aislado no provee información suficiente para describir efectivamente un Servicio Web. El esquema no describe los patrones de mensajes entre el cliente y el servidor. Por ejemplo, un cliente necesita conocer si tiene que esperar una respuesta cuando envía una orden. Un cliente también necesita conocer sobre que protocolo de transporte el Servicio Web espera recibir los requerimientos. Finalmente, el cliente necesita conocer la dirección donde el Servicio Web puede ser encontrado.

Esta información es provista por el **Web Service Description Language** (WSDL) ideado por Microsoft, IBM y otros. WSDL es un documento XML que describe un Servicio Web particular completamente. Herramientas tales como ASP.NET, WSDL.exe y Remoting SOAPUDS.exe pueden consumir WSDL y construir proxies automáticamente para el desarrollador.

Eligiendo el mecanismo de descubrimiento

Una vez que desarrollamos y documentamos un Servicio Web, ¿cómo podemos hacer para que potenciales clientes puedan localizarlo?. Si el Servicio Web está diseñado para ser consumido por un miembro de nuestro equipo de desarrollo, podemos hacerlo bastante informalmente compartiendo la URL o el documento WSDL. Pero cuando los potenciales clientes están sobre Internet esta es una historia enteramente diferente. [6]

Fue necesario un medio común para hacer público el Servicio Web. El **Universal Description, Discovery and Integration** (UDDI) provee justamente tal mecanismo. UDDI es un servicio de directorio centralizado estándar de la industria que puede ser usado para hacer público y localizar los Servicios Web. UDDI permite a los usuarios buscar Servicios Web por el nombre de la compañía, categoría y tipo de Servicio Web. [6]

Los Servicios Web pueden también hacerse públicos vía DISCO, un documento XML propietario definido por Microsoft que permite a los sitios Web hacer públicos los servicios que ellos exponen. DISCO define un protocolo simple facilitando un hipervínculo para ubicar recursos. El consumidor primario de DISCO es Microsoft Visual Studio .NET. Un desarrollador puede direccional un servidor Web particular y navegar a través de varios Servicios Web expuestos por el servidor. [6]

Lo que está ausente para los Servicios Web

Debemos estar advirtiendo que algunos ítems claves encontrados en la infraestructura de componentes distribuidos no están definidos para los Servicios Web. Dos de las omisiones más notables son las API's para crear y consumir Servicios Web y un conjunto de componentes de servicio tales como soporte para transacciones distribuidas. Veamos los siguientes puntos: [6]

- *API específica del Servicio Web*: la mayoría de la infraestructuras de componentes distribuidos definen una API para llevar a cabo tareas como inicialización en tiempo de ejecución, crear una instancia de un componente y recuperar la metadata usada para describir el componente. Debido a que la mayoría de los lenguajes de programación de alto nivel proveen algún grado de interoperabilidad con C, la API es usualmente expuesta como un conjunto de métodos C. RMI hasta el momento está estrechamente acoplado a su API con un único lenguaje de alto nivel, Java. Es un esfuerzo asegurar que los Servicios Web sean agnósticos del lenguaje de programación. [6]
- *Servicios de componentes*: la plataforma de Servicios Web no provee muchos de los servicios comúnmente encontrados en infraestructuras de componentes distribuidos, tales como manejo del tiempo de vida de los objetos, pooling de objetos y soporte para transacciones distribuidas. Algunos servicios, tales como soporte para transacciones distribuidas, pueden ser introducidos luego con la maduración de la tecnología. Otros, tales como pooling y la posibilidad de manejar el tiempo de vida de los objetos, pueden ser considerados como un

detalle de la implementación de la plataforma. Por ejemplo, .Net Remoting define extensiones para proveer soporte para manejar el tiempo de vida de los objetos y los Microsoft Component Services proveen soporte para el pooling de objetos. [6]

Cuándo usar Servicios Web

Explicaremos algunos escenarios donde podremos darnos cuenta de los beneficios de usar Servicios Web. [35]

- *Comunicación a través de firewalls:* cuando construimos una aplicación distribuida con miles o cientos de usuarios esparcidos en muchos lugares, siempre está el problema de comunicación entre el cliente y el servidor por los firewalls y los servidores proxy. Como con los Servicios Web podemos usar SOAP sobre HTTP no tendremos problemas con este hardware. [35]
- *Integración de aplicaciones:* conocemos que una buena parte del esfuerzo de desarrollo es consumido en integrar aplicaciones escritas en varios lenguajes y corriendo sobre sistemas dispares. Típicamente necesitamos recibir datos en nuestra aplicación desde otra corriendo sobre un mainframe IBM o necesitamos enviar datos desde nuestra aplicación hacia un mainframe o aplicación basada en Unix. Aún sobre la misma plataforma, aplicaciones de diferentes proveedores necesitan ser integradas. Entonces exponiendo alguna de sus funcionalidades y datos vía Servicios Web, una aplicación provee un mecanismo estándar para que otras aplicaciones se integren con ella. [35]
- *Integración Bussines-to-Bussines:* los Servicios Web facilitan la tecnología para la integración B2B. Usando Servicios Web, podemos exponer procesos de negocio vitales a proveedores y clientes autorizados. Por ejemplo, podemos exponer los procesos de órdenes de compra y facturación, por esta razón facilitamos a nuestros clientes enviar electrónicamente las órdenes de compra y a nuestros proveedores enviar electrónicamente sus facturas. La ventaja fundamental de usar Servicios Web para integración B2B es el bajo costo de interoperabilidad. exponiendo nuestras funciones de negocio como Servicios Web, facilitamos a cualquier parte autorizada invocar estos servicios independientemente de la plataforma y lenguaje de programación que ellos usen. Esto reduce el tiempo y el esfuerzo que toma integrar un proceso de negocio a través de compañías. [35]
- *Reuso de software:* el reuso de software puede tomar lugar de varias formas y a diferentes niveles. La forma más básica de reusar código es a través de reuso de código fuente o clases. Otra forma es el reuso basado en componentes binarios. Hoy en día hay un mercado significante de reuso de componentes de software tales como grillas y otros componentes de interfaces de usuario. Pero el reuso de software siempre ha estado limitado por un factor clave: podemos reusar el código pero no los datos. La razón de esto es que podemos distribuir fácilmente componentes o código fuente, pero no podemos distribuir fácilmente datos a menos que estos sean medianamente estáticos y que no se espere que cambien

mucho.
Los Servicios Web dejan reusar el código junto con los datos. En vez de comprar un componente de terceras partes e instalarlo y llamarlo desde la aplicación, nuestra aplicación llamará a un Servicio Web remoto. Por ejemplo, si los usuarios de nuestra aplicación entran direcciones de correo electrónico que queremos validar, podemos enviar éstas a validar al Servicio Web. [35]

Cuándo no usar Servicios Web

Esta explicación no estaría completa si no indicamos las situaciones donde no debemos usar Servicios Web. Está claro que los Servicios Web ofrecen el mayor beneficio en los casos donde se desea interoperabilidad y/o ejecución remota sobre la Web. [35]

- *Aplicaciones de una sola máquina:* aún hay muchas aplicaciones de escritorio para uso de negocios o personal. Algunas de estas aplicaciones necesitan comunicarse con otras corriendo sobre la misma máquina. En este caso, es casi siempre mejor usar una API nativa más que un Servicio Web. Tecnologías de componentes de software tales como COM y librerías de clases .NET o componentes son muy efectivos en estos escenarios porque usar tales componentes requiere relativamente poca sobrecarga. [35]
- *Aplicaciones homogéneas sobre una LAN:* en muchos casos, tendremos aplicaciones desarrolladas usando Visual Basic y/o C++ y COM y corriendo sobre plataformas Windows sobre la misma LAN. Por ejemplo, podemos tener dos aplicaciones servidor esperando para comunicarse con otros, más comúnmente clientes Win32 o Windows Forms. Es mucho más eficiente para estas dos aplicaciones comunicarse usando DCOM más que SOAP/HTTP. Similarmente, si tenemos una aplicación .NET que necesita comunicarse con otra sobre una LAN, podemos usar .NET Remoting. [35]

4.5. Simple Object Access Protocol (SOAP)

SOAP está en el corazón de los Servicios Web, proveyendo una manera estándar de empaquetar los mensajes. SOAP ha recibido atención en parte por la facilidad de comunicación al estilo RPC entre un cliente y un servidor remoto. Pero muchos protocolos han sido creados para facilitar la comunicación entre dos aplicaciones, incluyendo: Sun RPC, Microsoft DCE, Java RMI y CORBA ORPC. Así que, ¿por qué SOAP está recibiendo tanta atención? [6]

Una de las razones principales es que SOAP tiene un soporte de la industria increíble. SOAP es el primer protocolo de este tipo aceptado por prácticamente todas las principales compañías de software del mundo. Compañías que raramente cooperan unas con otras están concentrándose en torno a este protocolo. Algunas de las principales

compañías que están apoyando SOAP son Microsoft, IBM, Sun Microsystems, SAP y Ariba.

Estas son algunas de las ventajas de SOAP [6]:

- *No está estrechamente acoplado a un lenguaje.* SOAP no especifica una API, la implementación de la API es independiente del lenguaje de programación y de la plataforma.
- *No está estrechamente acoplado a un protocolo de transporte en particular.* La especificación SOAP describe cómo un mensaje SOAP debe ser unido a HTTP. Pero un mensaje SOAP no es más que un documento XML, así que puede ser transportado sobre cualquier protocolo que sea capaz de transmitir texto.
- *No está atado a ninguna infraestructura de objetos distribuidos.* La mayoría de los sistemas de objetos distribuidos pueden ser extendidos para soportar SOAP. Es importante darse cuenta que aún con SOAP, middlewares tales como COM+ juegan un rol importante en la empresa. Componentes middleware todavía son responsables de algunas de las características más avanzadas en el manejo de objetos. Estas características incluyen el manejo del tiempo de vida de los objetos, transacciones, pooling de objetos y pooling de recursos. SOAP permite la interoperabilidad entre diferentes sistemas que están usando componentes middleware de proveedores rivales.
- *Impulsa estándares existentes de la industria.* Los principales contribuidores a la especificación SOAP evitaron intencionalmente no reinventar nada. Ellos optaron por extender los estándares existentes para que cumplan con sus necesidades. Por ejemplo, SOAP impulsó XML para codificar los mensajes. También usaron su propio sistema de tipos, SOAP impulsó las definiciones de tipos ya definidas dentro de la especificación del XSD.
- *Permite la interoperabilidad a través de múltiples ambientes.* SOAP fue construido sobre los estándares existentes de la industria, así aplicaciones que corren sobre plataformas que soportan estos estándares pueden comunicarse efectivamente vía mensajes SOAP con aplicaciones corriendo sobre otras plataformas. Por ejemplo, una aplicación de escritorio corriendo sobre una PC puede comunicarse efectivamente con una aplicación que es capaz de enviar y recibir mensajes XML sobre HTTP corriendo sobre un mainframe.

Anatomía de un mensaje SOAP

La clave en el aspecto arquitectural de SOAP es su diseño simple con el que se pretende alentar a los proveedores a adoptar e implementar el protocolo. SOAP no trata de resolver todos los problemas de comunicación en aplicaciones distribuidas, en lugar de eso se enfoca sobre los estándares mínimos requeridos para enviar mensajes desde una aplicación a otra. Por ejemplo, SOAP no incluye especificaciones para seguridad o de transacciones distribuidas. Sin embargo, la arquitectura de SOAP es flexible y extensible.

La arquitectura de SOAP se centra en enviar un mensaje SOAP desde el emisor (el cliente) a su último destino (el Servicio Web) con opcionalmente nodos intermedios entre los dos. Notemos que no hay nada en esta arquitectura que hable sobre acceso a objetos o sobre invocar métodos remotamente como en RPC. En cambio, SOAP se enfoca en enviar un mensaje desde el emisor al receptor [35].

SOAP provee una forma estándar de empaquetar un mensaje. Un mensaje SOAP está compuesto de una envoltura (envelope) que contiene un encabezado opcional y el cuerpo del mensaje que es mandatario. El cuerpo contiene la carga útil (los datos a enviar) y/o información de error opcional. La siguiente figura muestra la arquitectura de un mensaje SOAP [6].

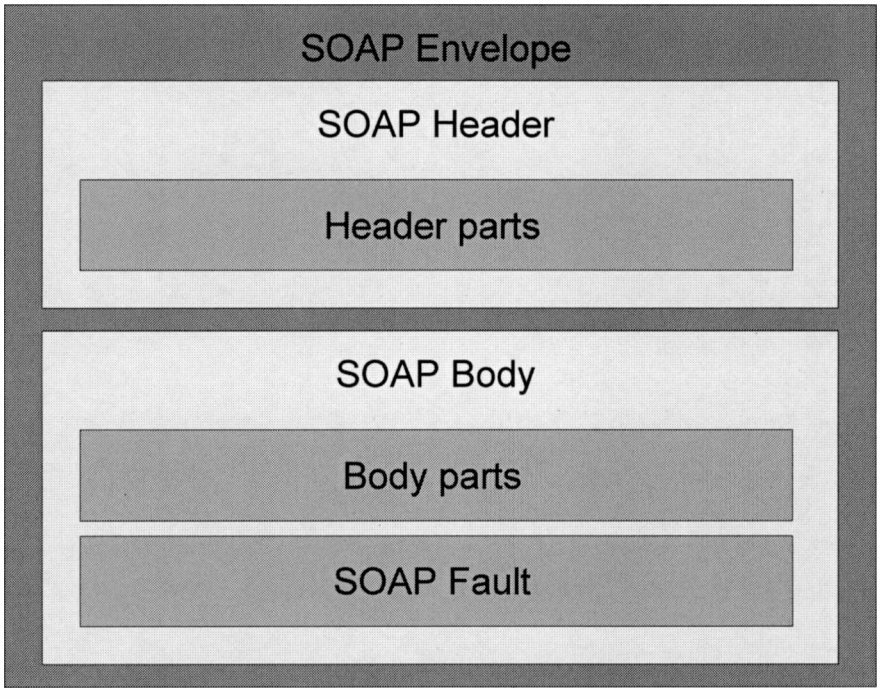


Figura 20. Mensaje SOAP.

El elemento raíz del documento es el elemento *Envelope* que contiene dos subelementos, el elemento *Body* y el elemento *Header*.

En la envoltura el elemento *Header* es opcional y contiene información sobre el mensaje. En el ejemplo de la figura 21, el *Header* contiene dos elementos describiendo al emisor del mensaje y al receptor deseado del mismo.

La envoltura debe contener un elemento *Body*. El *Body* contiene la carga útil del mensaje. En el ejemplo de la figura 21, el *Body* contiene una cadena de caracteres.

Como muestra el ejemplo de la figura 21 cada elemento específico de SOAP tiene el espacio de nombres *soap* como prefijo. Este prefijo está definido dentro del elemento *Envelope* y apunta al esquema SOAP que describe la estructura de un mensaje. El prefijo es agregado a cualquier elemento definido dentro del espacio de nombres SOAP. Estos elementos están completamente reservados. El prefijo *soap* indica que el elemento *Envelope* es una instancia del tipo *Envelope* de SOAP [6].

El último elemento del mensaje es el elemento *Fault* que es usado para comunicar información de error [35].

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header>
        <!--Optional header information goes here-->
        <To>Scott</To>
        <From>Suzanne</From>
    </soap:Header>
    <soap:Body>
        <!--Message goes here -->
        Please pick up some milk on your way home from work
    </soap:Body>
<soap:Envelope>
```

Figura 21. Mensaje SOAP.

Actores SOAP

Un *actor* (actor) SOAP es cualquiera que actúe sobre el contenido del mensaje. Hay dos tipos de actores, *default actors* (actores por defecto) e *intermediaries actors* (actores intermediarios). [6]

El actor por defecto es el receptor final deseado del mensaje SOAP. Un intermediario recibe un mensaje SOAP y puede actuar sobre el (incluso modificarlo de alguna manera) antes de reenviarlo, como muestra la siguiente figura. Aunque los intermediarios puedan modificar los datos transferidos desde el cliente hacia el actor por defecto, el mensaje es considerado el mismo.

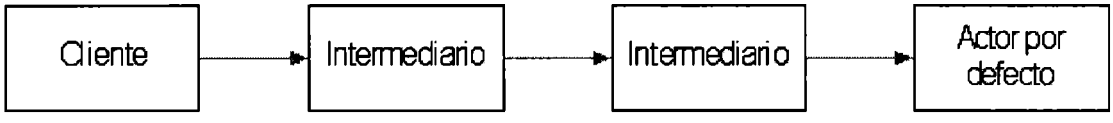


Figura 22. Actores SOAP.

El elemento Header

SOAP sólo provee la funcionalidad mínima requerida para invocar Servicios Web y dejar a los implementadotes extenderlo libremente de la forma necesaria más conveniente para el cliente y el servicio [35]. El elemento *Header* es una manera de

extender la funcionalidad de SOAP para comunicar información que no es necesariamente parte de los datos del mensaje pero es necesaria para procesar el mensaje. Por ejemplo, si el actor por defecto recibe un mensaje en el que el cuerpo está comprimido, necesitará conocer que tipo de algoritmo de compresión fue usado para descomprimir el mensaje. Incluir información sobre el algoritmo de compresión dentro del cuerpo del mensaje no es sensato debido a que el cuerpo mismo está comprimido, ubicar este tipo de información en el encabezado del mensaje es más apropiado [6].

Algunos otros usos del encabezado pueden ser [6]:

- *Autenticación.* El receptor puede requerir el emisor para autenticarlo por sí mismo antes que el mensaje pueda ser procesado.
- *Información de seguridad.* Si el receptor necesita asegurar que el contenido del mensaje no ha sido alterado, el emisor puede firmar digitalmente el cuerpo del mensaje y ubicar el resultado dentro del encabezado.
- *Información de direccionamiento.* Si el mensaje necesita ser direccionado a muchos destinos, los destinos y su orden pueden ser incluidos en el encabezado.
- *Transacciones.* El receptor puede tener que ejecutar alguna acción en el alcance de la transacción del emisor.
- *Información de pago.* Si el receptor del mensaje provee servicios a los clientes basándose en una tarifa "pague por uso", la información necesaria de pago puede ser incluida en el encabezado.

El elemento *Header* es un elemento opcional. Los encabezados pueden ser enviados con los mensajes de requerimiento y de respuesta para comunicar información al servicio o al cliente respectivamente [35].

El elemento *Header* puede ser agregado como un elemento hijo inmediato dentro del elemento *Envelope*. Las entradas en el encabezado aparecen como nodos hijos dentro del elemento *Header*. Por ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
      <soap:Header>
            <Digest>B839D234A3F87</Digest>
      </soap:Header>
      <soap:Body>
            <StockReport>
                  <Symbol>MSFT</Symbol>
                  <Price>74.53</Price>
            </StockReport>
      </soap:Body>
</soap:Envelope>
```

El mensaje SOAP contiene un elemento *Digest* en el encabezado que la aplicación remota puede usar para asegurar que los mensajes no han sido alterados.

Atributo *mustUnderstand*

Debido a que el encabezado es opcional, el receptor del mensaje puede elegir ignorarlo. Sin embargo, alguna información que pueda ser incluida en el encabezado no debe ser ignorada por el receptor deseado. Si el encabezado no es entendido o no puede ser manejado apropiadamente, la aplicación puede no funcionar apropiadamente. Por lo

tanto, necesitamos una forma de distinguir entre la información del encabezado que es informativa de la que es crítica. [6]

Podemos especificar si el receptor del mensaje debe entender un elemento en el encabezado con el atributo *mustUnderstand* con el valor en *1* en el elemento. El siguiente ejemplo actualiza la información de la cuenta del usuario dentro del alcance de la transacción: [6]

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
      <soap:Header>
            <TransactionId soap:mustUnderstand="1">123</TransactionId>
      </soap:Header>
      <soap:Body>
            <UpdateAccountInfo>
                  <email>sshort@microsoft.com</email>
                  <firstName>Scott</firstName>
                  <lastName>Short</lastName>
            </UpdateAccountInfo>
      </soap:Body>
</soap:Envelope>
```

Atributo *actor*

Un mensaje SOAP puede ser direccionado a través de muchos intermediarios antes de alcanzar su destino final. Por ejemplo, el documento anterior puede ser direccionado a través de un intermediario responsable de crear un contexto de transacción. En este caso, podemos querer especificar claramente que el elemento *TransactionId* del encabezado debe ser procesado por el intermediario generador de la transacción antes que por el actor por defecto. [6]

La especificación SOAP provee el atributo *actor* para marcar determinados encabezados SOAP para que los procesen ciertos intermediarios. El valor de este atributo es la *Uniform Resource Identifier* (URI) del intermediario que se desea que use el encabezado. Si se desea que el encabezado sea procesado por el siguiente intermediario que reciba el mensaje, el atributo *actor* puede tener el valor *http://schemas.xmlsoap.org/soap/actor/next*. Por otra parte al atributo *actor* se le puede colocar la URI que identifique a un intermediario específico. Por ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
      <soap:Header>
            <TransactionId soap:mustUnderstand="1"
            actor="urn:TransactionCoordinator">123</TransactionId>
      </soap:Header>
      <soap:Body>
            <TransferFounds>
                  <Source>804039836</Source>
                  <Destination>804039836</Destination>
                  <Amount>151.43</Amount>
            </TransferFounds>
      </soap:Body>
</soap:Envelope>
```

Debido a que el elemento *TransactionId* del encabezado es deseado por el intermediario coordinador de transacciones, su atributo *actor* está con la URI de este intermediario. El atributo *mustUnderstand* esta puesto de tal forma que si el

intermediario coordinador de transacciones no entiende el elemento *TransactionId* del encabezado, este debe disparar un error.

Si el mensaje es pasado a otro receptor, cualquier elemento del encabezado especificado para el intermediario debe ser eliminado antes que el mensaje sea reenviado. El intermediario puede, sin embargo, agregar elementos de encabezado adicionales antes de reenviar el mensaje al siguiente receptor. En este ejemplo, el intermediario coordinador de transacciones debe remover el elemento antes de reenviar el mensaje a la aplicación de facturación. [6]

Un punto importante es que direccionando el mensaje directamente al actor por defecto no es considerado un error. Colocar el atributo *mustUnderstand* con el valor *1* en combinación con el atributo *actor* con *urn:TransactionCoordinator* no asegura que el mensaje será direccionado a través del intermediario. Esto solo significa que si el mensaje alcanza el intermediario coordinador de transacciones, este deberá comprender la entrada *TransactionId* del encabezado o disparar un error.

El elemento Body

Un mensaje SOAP válido debe tener un elemento *Body*. El *Body* contiene la caga útil del mensaje. No hay restricciones sobre como el cuerpo puede ser codificado. El mensaje puede ser una simple cadena de caracteres, un arreglo de bytes codificado o XML. El único requerimiento es que el contenido no puede tener ningún carácter que invalide el documento XML resultante.

Los mensajes SOAP pueden generalmente ser ubicados en dos categorías: mensajes orientados al proceso (RPC) y mensajes orientados al documento.

- *RPC:* Los mensajes orientados al proceso proveen dos vías de comunicación y se refieren comúnmente a mensajes de llamadas a procedimientos remotos (Remote Procedure Call). El cuerpo de un mensaje RPC contiene información sobre la acción requerida del servidor y cualquier parámetro de entrada y salida [6]. El elemento *Body* contiene el nombre del método o procedimiento remoto invocado y un elemento por cada parámetro del procedimiento [35].
- Los mensajes orientados al documento generalmente facilitan comunicación de una vía. Documentos de negocio tales como órdenes de compra son ejemplo de mensajes orientados al documento [6]. El elemento *Body* contiene uno o más elementos hijos llamados *parts* (partes). No hay reglas SOAP para el contenido del *Body*, este puede contener cualquier cosa que el emisor y el receptor hayan acordado [35].

Además, para los dos estilos de mensajes SOAP, hay dos formatos para serializar datos dentro en XML. El formato elegido determina como los datos son serializados dentro de los elementos *Body* y *Header* [35]. Esto es una buena idea conforme a que establece un esquema de codificación tal que este permita al emisor interoperar más fácilmente con el receptor usando un conjunto bien conocido de reglas de serialización [6]. Las definiciones de cada formato de serialización son [35]:

- *Codificación:* Los datos son serializados de acuerdo a algunas reglas de serialización. Las más comunes son las especificadas en la especificación SOAP.
- *Literal:* Los datos son serializados de acuerdo a un XML Schema, usualmente XSD. No hay reglas de codificación especiales que indiquen como serializar los datos. Con este formato, el cliente y el servicio tratan los datos en términos de documentos XML más que en términos de objetos y estructuras.

Teóricamente, la elección de usar RPC o documentos es independiente de la elección del formato de serialización, dando un total de cuatro combinaciones diferentes.

Sin embargo, prácticamente, todas las implementaciones tienden a combinar RPC con codificación y documentos con el formato literal. Esto da significado a la división de las aplicaciones en dos categorías: aquellas que intercambian documentos de negocio, tales como aplicaciones B2B, que tienden a usar mensajes SOAP documento/literal y aquellas que usan SOAP como un protocolo RPC para invocar objetos remotos que tienden a usar mensajes SOAP RPC/codificación [35].

Dos mensajes SOAP son aparejados para facilitar una llamada a un método RPC con SOAP: el mensaje de requerimiento y el correspondiente mensaje de respuesta. Información sobre el método objetivo junto con cualquier parámetro de entrada es pasado al servidor vía un mensaje de requerimiento. El servidor evoca entonces algún comportamiento pedido del cliente y retorna los resultados y cualquier parámetro de retorno.

Un documento de negocio tal como una orden de compra o una factura puede ser codificado dentro del cuerpo de un mensaje SOAP y direccionado a su receptor. El receptor del documento puede o no enviar un mensaje de acuso de recibo de vuelta al emisor [6].

Elemento *fault*

No siempre todo va como lo planeamos, a veces el servidor encontrará un error mientras procesa el mensaje del cliente. SOAP provee una forma estándar de comunicar mensajes de error de vuelta al cliente.

Independientemente del estilo que haya sido usado para crear el mensaje, la especificación SOAP tiene acordada la forma de organizar los datos para reportar un error. El cuerpo del mensaje debe contener un elemento *Fault* con la siguiente estructura: [6]

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
      <soap:Body>
            <soap:Fault>
                  <soap:faultcode>Client.Security</soap:faultcode>
                  <soap:faultstring>Access denied.</soap:faultstring>
                  <soap:faultactor>http://abc.com</soap:faultactor>
                  <soap:detail>
                        <MyError>
                              <Originator>File System</Originator>
                              <Resource>MySecureFile.txt</Resource>
                        </MyError>
                  </soap:detail>
            </soap:Fault>
```

```
    </soap:Body>
</soap:Envelope>
```

El elemento *faultcode* contiene un valor que es usado para determinar programáticamente la naturaleza del error. La especificación SOAP define un conjunto de códigos de error que podemos usar para describir errores SOAP básicos. Los códigos de error están listados en la tabla siguiente. [6]

Código de Falla	Descripción
VersionMismatch	Fue especificado un espacio de nombres inválido para el elemento *Envelope*
MustUnderstand	Un elemento hijo inmediato dentro del encabezado SOAP contiene un atributo *mustUnderstand* con el valor *1* no fue entendido o acatado por el servidor
Client	El contenido del mensaje es la causa del error. Posibles causas de errores *Client* incluyen mensajes mal formados o información incompleta en los mensajes
Server	La causa del error no es directamente atribuible al contenido del mensaje. Ejemplos de este tipo de error incluyen que el servidor no sea capaz de obtener los recursos apropiados para procesar el mensaje (tales como conexiones de base de datos) o errores lógicos durante el procesamiento del mensaje

Podemos agregar más códigos de error específicos a los códigos de error SOAP listados en la tabla usando la notación puntual y ordenando los códigos de error desde el menos específico al más específico. Por ejemplo, si el servidor no es capaz de abrir una conexión a la base de datos que es requerida para procesar el mensaje del cliente, puede ser generado el siguiente código de error:

```
<faultcode>Server.Database.Connection</faultcode>
```

Debido a que el error no fue resultado directo del mensaje del cliente, el código de falla base es *Server*. Un código de error más descriptivo es agregado al final del código de error base.

El elemento *faultstring* debe contener una cadena de caracteres humanamente legible que describa el error encontrado.

El elemento *detail* es opcional y es donde podemos poner cualquier información de error o más elementos hijos. Estos elementos hijos están definidos a nivel de aplicación. Este elemento nos facilita la comunicación de errores en tiempo de ejecución o de información de excepciones [35].

Podemos usar opcionalmente el elemento *faultactor* que indica el origen exacto del error. La única excepción es si un intermediario generó el error. Si el error fue generado por cualquier punto distinto del receptor final del mensaje SOAP, el elemento *faultactor* debe contener la URI que identifica al origen del error. En otro caso, la URI puede ser omitida.

Usando mensajes SOAP RPC

Una de las metas de diseño originales de SOAP fue proveer una forma abierta y estándar para facilitar RPC usando tecnologías Internet tales como XML y HTTP.

Para facilitar el comportamiento requerimiento-respuesta necesario para RPC, necesitamos dos mensajes SOAP: uno para el requerimiento y otro para la respuesta. Así es como el mensaje del requerimiento deberá codificar una función C# que agrega dos números: [6]

```
public int Add (int x, int y)
{
      return x + y;
}
```

El método *Add* acepta dos enteros como parámetros de entrada y pasa de vuelta al cliente el resultado como parámetro de retorno. Los parámetros de entrada deben ser empaquetados dentro del cuerpo del requerimiento de forma que puedan ser enviados a la aplicación objetivo. Este es el mensaje resultante para el requerimiento para *Add*: [6]

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
      <soap:Body>
            <Add>
                  <x>1</x>
                  <y>2</y>
            </Add>
      </soap:Body>
</soap:Envelope>
```

El elemento *Body* contiene un elemento *Add*. Cada uno de los parámetros de entrada está representado como un subelemento dentro del elemento *Add*. El orden de los elementos *x* e *y* debe coincidir con el orden en el que los parámetros son especificados en la firma del método. En otras palabras, ubicar *y* antes que *x* es inválido.

Protocolo de enlace

Necesitamos una manera de enviar el mensaje a una aplicación remota. Una ventaja de SOAP es que no está atado a un protocolo de transporte en particular. Los mensajes SOAP pueden ser enviados sobre cualquier protocolo de transporte que sea capaz de llevar XML.

Cuando un mensaje SOAP es llevado por un protocolo de transporte particular en conocido como *protocolo de enlace (protocol binding)*. Un protocolo de enlace puede ser definido para explotar cualquier característica única del protocolo de transporte.

A pesar del enorme éxito de HTTP como el protocolo de aplicación de Internet, éste está limitado a comandos bastante simples centrados en requerir y responder recursos. El resultado es que hoy en día tenemos millones de computadoras interconectadas para principalmente navegar la Web pero no pueden, a pesar de la conectividad, intercambiar libremente datos entre aplicaciones. SOAP propone resolver

este problema definiendo un protocolo estándar que cualquier aplicación pueda usar para comunicarse e intercambiar datos con cualquier otra aplicación. La figura siguiente muestra como SOAP puede ser usado sobre TCP/IP dando empuje a la infraestructura de Internet actual [35].

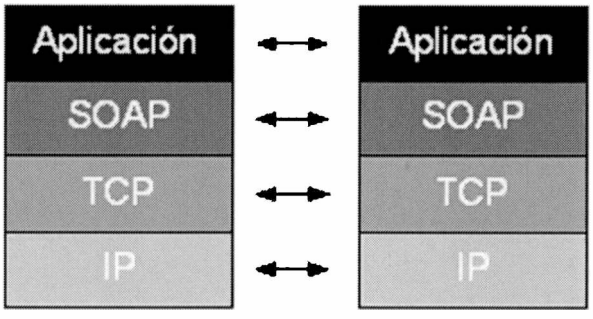


Figura 23. SOAP sobre TCP/IP.

SOAP es un protocolo a nivel de aplicación entonces puede trabajar directamente sobre un protocolo de transporte como TCP. Sin embargo, la infraestructura de Internet de hoy en día cuenta con proxies y firewalls que típicamente permiten solo tráfico HTTP.

Por esto, para que todas las aplicaciones conectadas a Internet se comuniquen, SOAP debe ser capaz de correr sobre la infraestructura actual de Internet incluyendo firewalls y proxies. Para alcanzar esto, SOAP puede ser sostenido sobre HTTP como muestra la siguiente figura.

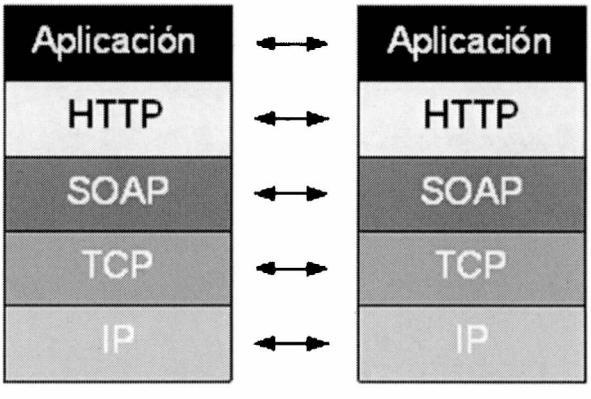


Figura 24. SOAP sobre HTTP.

Sostener SOAP sobre HTTP significa que los mensajes SOAP son enviados como parte de un requerimiento o de una respuesta haciendo fácil comunicarse sobre cualquier red que permita tráfico HTTP. HTTP es también una buena elección porque, como los navegadores Web, éste es penetrante sobre todas las plataformas y dispositivos [35].

Discutidamente, el protocolo de transporte más popular usado para enviar mensajes SOAP es HTTP. Sin embargo, pueden también ser enviados vía SMTP, vía fax, etc.

La especificación SOAP describe sólo un protocolo de enlace: enviar mensajes SOAP vía HTTP POST.

La mayoría de las implementaciones SOAP, incluyendo .NET, soportan el protocolo HTTP. Debido a que la mayoría de los sistemas soportan HTTP, éste se ha convertido discutiblemente en el protocolo elegido para asegurar que un Servicio Web tenga un alto grado de interoperabilidad entre diferentes plataformas. Entre las ventajas del protocolo HTTP se incluyen las siguientes:

- *Es amigable con los firewalls.* Protocolos antiguos como DCOM no lo son. La mayoría de los firewalls tienen el puerto 80, como mínimo, abierto para el tráfico HTTP.
- *Tiene un soporte de infraestructura robusto.* Muchas tecnologías han sido introducidas en el esfuerzo por incrementar la escalabilidad y disponibilidad de aplicaciones basadas en HTTP.
- *Es esencialmente sin estados.* La naturaleza sin estado de HTTP ayuda a asegurar que la comunicación entre el cliente y el servidor sea confiable, especialmente a través de Internet. Las caídas intermitentes en la conexión trae problemas a protocolos como DCOM y CORBA.
- *Es simple.* El protocolo HTTP está compuesto de una sección de encabezado y una sección cuerpo.
- *Se corresponde bastante bien para intercambiar mensajes estilo RPC.* HTTP es un protocolo natural para comunicaciones al estilo RPC porque un requerimiento es siempre acompañado de una respuesta.
- *Está expandido.* Prácticamente todos los sistemas de red soportan HTTP.

Un requerimiento HTTP está compuesto de dos partes, un cuerpo y un encabezado. El encabezado contiene información sobre el requerimiento y sobre el cliente que lo envía. El cuerpo sigue al encabezado y está delimitado por dos pares de carriage-return/linefeed. El cuerpo contiene la carga útil, que en este caso será un mensaje SOAP. El siguiente es un ejemplo de un requerimiento HTTP que contiene un mensaje SOAP:

```
POST /SomeWebService HTTP/1.1
Content-Type: text/xml
SOAPAction: "http://somedomain.com/SomeWebService.wsdl"
Content-Length: 243
Host: sshort3


<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
            xmlns:soap-enc="http://schemas.xmlsoap.org/soap/encodign/">
      <soap:Body>
            <Add>
                  <x>2</x>
                  <y>2</y>
            </Add>
      </soap:Body>
</soap:Envelope>
```

El encabezado HTTP para un mensaje SOAP es similar al de un requerimiento HTML, con un par de diferencias: la entrada del encabezado *Content-Type* tiene

siempre el valor *text/xml*, y la otra diferencia es que todo requerimiento HTTP POST SOAP debe contener la entrada de encabezado *SOAPAction*.

SOAPAction es usada para comunicar el objetivo del mensaje SOAP. La URI puede ser representada en cualquier formato y no es requerido que sea resoluble.

4.6. XML Schema

SOAP provee un método estándar de codificación de datos dentro de un documento XML. Esta tecnología es también extremadamente flexible. Cualquier cosa puede ser codificada dentro del cuerpo de un mensaje SOAP siempre y cuando ésta no invalide el XML. El cuerpo de un mensaje puede contener un requerimiento sobre la última información sobre el tiempo, una orden de compra, una imagen satelital, etc. [6]

Con la variedad de contenido y de tipos de datos que pueden estar contenidos dentro de un mensaje SOAP, necesitamos una forma de expresar la estructura de un mensaje. También necesitamos una manera de determinar los tipos de datos que deben aparecer dentro del mensaje. [6]

Una solución potencial para el desarrollador puede ser proveer un ejemplo de un mensaje SOAP que sea válido. Por ejemplo, supongamos que necesitamos interactuar con un Servicio Web para ubicar una orden para un proveedor, y el proveedor nos provee el siguiente mensaje de ejemplo: [6]

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
      <soap:Body>
            <PurchaseItem>
                  <Item>Apple</Item>
                  <Quantity>12</Quantity>
            </PurchaseItem>
      </soap:Body>
</soap:Envelope>
```

El documento XML es bastante directo. Del mensaje de ejemplo, podemos ver que el Servicio Web acepta dos parámetros, *Item* y *Quantity*, ambos son elementos hijos del elemento *PurchaseItem*. [6]

El problema es que el mensaje de ejemplo deja una cantidad de cosas que permanecen ambiguas. Por ejemplo, conocemos la necesidad de pasar el parámetro *Item*, pero ¿debe contener la descripción del ítem?, ¿debe contener el identificador del ítem?, ¿este valor está limitado a un número máximo de caracteres?

También hay muchas preguntas relativas al parámetro *Quantity*. ¿Podemos especificar cantidades como 1.5?, ¿hay un mínimo o máximo de compra?

Una forma de aclarar esta ambigüedad es hacer que el mensaje de ejemplo sea acompañado por un documento que describa todos los matices del mensaje. El Servicio Web necesita validar el mensaje recibido. También, probablemente queramos validar el mensaje antes de ser enviado al Servicio Web.

Lo que estamos necesitando es una forma estándar de describir la estructura y el tipo de información que debe ser contenida dentro de un mensaje XML enviado al Servicio Web. En otras palabras, necesitamos una manera de representar el esquema que

un documento XML debe cumplir para ser procesado por el Servicio Web. Además, el esquema necesita ser estandarizado y estar acompañado por un conjunto de API's que puedan ser usadas para validar programáticamente el documento XML contra el esquema. [6]

Describiendo documentos XML

Podemos usar un esquema para describir la estructura de un documento XML y su información de tipo. Las dos tecnologías dominantes para definir un esquema XML son *Document Type Definitions* (DTDs) y *XML Schema* (XSD). Podemos usar DTDs para definir la estructura de un documento XML pero no para describir los contenidos de un documento. Aquí hay un ejemplo: [6]

```
<!—Request Message -->
<!ELEMENT PurchaseItem        (Quantity, Item)>
      <!ELEMENT Quantity      (#PCDATA)>
      <!ELEMENT Item          (#PCDATA)>

<!-- Response Message -->
<!ELEMENT PurchaseItemResult  (Amount)>
      <!ELEMENT Amount        (#PCDATA)>
```

De un primer vistazo debemos ver que la sintaxis DTD no está basada en XML. Las DTDs no pueden ser analizadas usando analizadores de XML y no pueden ser fácilmente incorporadas dentro de documentos XML. Veremos por qué esto es importante cuando veamos WSDL.

Las DTDs describen la estructura de los documentos pero no pueden expresar los tipos de datos que ésta contiene. No hay una noción fundamental de tipos dentro de DTDs tal como *integers* o *strings*, ni soportan la definición de nuestros propios tipos. [6]

En el ejemplo anterior, los elementos *Quantity* e *Item* son declarados como *#PCDATA*, esto no da ningún indicio sobre el tipo de dato que pueden contener. Por ejemplo, la DTD no indica si son válidas cantidades como 1.5. Esta tampoco indica si el elemento *Item* debe contener el identificador numérico del producto o sólo una cadena de caracteres con la descripción del ítem.

Un estándar propuesto por la W3C llamado *Datatypes for DTDs* (DT4DTD) 1.0 (http://www.w3.org/TR/dt4dtd) provee una forma de inyectar información de tipos dentro de esquemas DTD. Las DTDs deben ser consideradas como la tecnología para definir los XSD porque sus limitaciones y carencias fueron subsanadas en estos. [6]

La manera recomendada para expresar esquemas para Servicios Web basados en XML es a través de XSD. Este comprende dos especificaciones manejadas por la W3C, XML Schema Parte 1: Estructuras (http://www.w3.org/TR/xmlschema-1/) y XML Schema Parte 2: Tipos de datos (http://www.w3.org/TR/xmlschema-2). [6]

XSD provee una rica sintaxis para definir esquemas usados para validar documentos XML. Este no sólo nos permite definir la estructura de un documento XML sino que también nos permite definir el tipo de dato que el documento contiene y cualquier restricción sobre los datos. También, nos permite especificar claves foráneas y restricciones de integridad referencial. Aquí hay un ejemplo:

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">

      <!-- Response Message (work - in - progress) -->
      <element name="Amount"/>

</schema>
```

Como podemos ver, el esquema de ejemplo es un documento XML válido que puede ser consumido por cualquier analizador estándar de XML. Una definición de esquema está contenida dentro de un elemento raíz *schema*. El esquema de ejemplo define un elemento llamado *Amount*.

Los documentos XML que pueden ser validados contra un esquema son llamados *instance documents* (documento de la categoría de ese esquema). El siguiente es un instance document para el esquema descrito anteriormente:

```
<?xml version="1.0"?>
<Amount>351.43</Amount>
```

Sistema de tipos

Tipos de datos integrados

Una de las características más útiles de XSD es que define un conjunto base de tipos de datos. Incluye tipos de la programación básica tales como *string*, *int*, *float* y *double*; tipos matemáticos tales como *integer* y *decimal*; y tipos XML tales como *NMTOKEN* e *IDREF*. [6]

Una de las ventajas más significantes del sistema de tipos de XSD es que es independiente de la plataforma. Los valores de los tipos están representados consistentemente sin importar el hardware, sistema operativo o software para procesamiento de XML usado. El sistema de tipos de XSD permite protocolos basados en XML tales como SOAP para alcanzar una fuerte interoperabilidad en ambientes de computación heterogéneos. [6]

Los tipos de datos son útiles para definir esquemas que describan el tipo de dato que debe estar contenido dentro del documento.

Anteriormente creamos un mensaje SOAP para enviar un requerimiento *PurchaseItem*. Ahora usaremos los tipos de datos integrados para indicar los tipos de parámetros:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
            xmlns:xsi="http://www.w3.org/201/XMLSchema-instance">
      <soap:Body>
            <PurchaseItem>
                  <Item xsi:type="xsi:string">Apple</Item>
                  <Quantity xsi:type="xsi:int">12</Quantity>
            </PurchaseItem>
      </soap:Body>
</soap:Envelope>
```

Agregamos un atributo *xsi:type* para cada elemento parámetro. El valor del atributo representa el tipo de dato del parámetro codificado. Acompañar los elementos dentro de un documento XML con información del tipo elimina todas las ambigüedades relativas al tipo de dato que el emisor codifica dentro del mensaje. El receptor del mensaje precedente conocerá que *Quantity* es representado como un *int* e *Item* es representado como un *string*.

La especificación SOAP define un *polymorphic accessor* como un elemento cuyo tipo es determinado en tiempo de ejecución. El polymorphic accessor es conceptualmente similar al tipo *Object* de C#. Si un polymorphic accessor aparece dentro de un mensaje SOAP, éste debe contener un atributo *type* indicando el tipo de dato del contenido del elemento.

Sistema de tipos de XSD

Los tipos simples representan valores escalares o no estructurados. Por ejemplo *int*, *date* y *string* son tipos simples. Los tipos complejos representan datos estructurados. Una clase o estructura puede ser representada como un elemento que contiene elementos hijos o atributos. En general, los elementos XML pueden tener atributos y pueden contener texto simple y/o elementos hijos. Si un elemento contiene atributos y/o elementos hijos, entonces es de un tipo complejo. Sin embargo, un elemento que no tiene atributos y contiene sólo texto es de un tipo simple porque representa a un valor escalar. Por otro lado, los atributos XML sólo contienen texto, por lo que siempre contienen valores escalares y son de tipos simples. [35]

El sistema de tipos de XSD es similar al sistema de tipos de .NET en que todo tipo deriva de algún otro tipo base. En .NET, *Object* es la raíz del sistema de tipos. Similarmente, el sistema de tipos de XSD es una jerarquía de tipos con el tipo integrado *anyType* como su raíz. Todo tipo, tanto integrado como definido por el usuario, deriva de algún otro tipo. [35]

De *anyType* derivan todos los tipos complejos y *anySimpleType,* los tipos que derivan directamente de este último son llamados tipos primitivos. Todos los otros tipos son llamados tipos derivados. Hay tres formas de derivar desde un tipo base: restricción, extensión y lista. Derivar por restricción es cuando un tipo deriva de un tipo base para restringir la definición del tipo base. El tipo NMTOKENS deriva de NMTOKEN para listar, esto significa que valores del tipo NMTOKENS son simplemente listas de valores del tipo NMTOKEN. Los tipos integrados no contienen ningún tipo derivado por extensión. Sin embargo, en la práctica, podemos necesitar definir nuevos tipos que extiendan al tipo base. [35]

Escribiendo esquemas XSD

Hay típicamente dos pasos para crear esquemas XSD. Definir nuevos tipos de datos y declarar elementos y atributos usando los tipos definidos así como los tipos integrados. [35]

Declarando elementos

Cuando declaramos un elemento especificamos su nombre y su tipo. Por ejemplo, si nuestro documento XML contiene un elemento llamado *quantity* que es un entero, lo declararíamos así:

```
<element name="quantity" type="int"/>
```

Esto dice que el documento debe contener exactamente un elemento llamado *quantity*. Podemos usar los atributos *minOccurs* y *maxOccurs* para cambiar el número de ocurrencias del elemento. Por defecto la cantidad de ocurrencias es *minOccurs=1*. El tipo del elemento puede ser un tipo integrado o un tipo definido por nosotros.

Definiendo tipos simples

Aunque hay muchos tipos simples integrados, aún necesitaremos definir nuevos tipos simples. Por ejemplo, si tenemos el método:

```
public void GetCustomerByName (string custName)
```

Donde, basándonos en el esquema de base de datos de la aplicación, *custName* es un *string* con una longitud máxima de 50 caracteres. En este caso no es suficiente usar el tipo integrado *string* como tipo de dato para *custName* porque no dice nada sobre su longitud máxima. En lugar de eso, definimos un nuevo tipo que derive de *string* y restrinja la longitud máxima a 50:

```
<simpleType name="LimitedLenString">
      <restriction base="string">
            <maxLength value="50"/>
      </restriction>
</simpleType>
```

El elemento *restriction* indica que este tipo simple deriva de *string* y lo restringe. Este elemento contiene uno o más elementos hijos llamados facetas (facets), que son usados para restringir al tipo base.

La siguiente tabla indica todas las facetas disponibles y una pequeña descripción de cada una: [35]

Faceta	Descripción
enumeration	Especifica un conjunto de valores permitidos
fractionDigits	El número máximo de dígitos en la parte fraccionaria. Aplica a tipos de datos derivados de *decimal*
length	Fija la longitud de los valores de este tipo. Note que la unidad de longitud depende del tipo base. Para los tipos derivados por lista, la longitud es el número de ítems en la lista
maxExclusive	Es el límite superior exclusivo, los valores deben ser menores que este límite
maxInclusive	Es el límite superior inclusivo, los valores deben ser menores o iguales a este límite
maxLength	La máxima longitud permitida. La unidad depende del tipo de dato
minExclusive	Es el límite inferior exclusivo, los valores deben ser mayores que este límite
minInclusive	Es el límite inferior inclusivo, los valores deben ser mayores o iguales a este límite
minLength	La mínima longitud permitida. La unidad depende del tipo de dato

pattern	Una expresión regular que restringe los valores al patrón específico
totaDigits	El número máximo de dígitos. Aplica a tipos derivados de *decimal*
whiteSpace	Debe ser: *preserve*, *replace* o *collapse*. Indica como los espacios en blanco son tratados. *Preserve* indica que todos los espacios en blanco deben ser dejados como están. *Replace* indica que cada tabulador, retorno de carro, y nueva línea sean reemplazados por un espacio en blanco. *Collapse* indica que el primer espacio en blanco es tratado como *replace*, los espacios contiguos son reemplazados por un espacio y el resto de los espacios son removidos

Definiendo tipos complejos

Los tipos complejos son usados para representar clases, estructuras, arreglos y otros datos estructurados.

```
<complexType name="exampleType">
      <sequence>
            <element name="elem1" type="string"/>
            <element name="elem2" type="string"/>
      </sequence>
</complexType>
```

El elemento *sequence* indica que los elementos de este tipo deben contener una secuencia de *elem1* seguido por *elem2*. Todos los elementos declarados dentro de un elemento *sequence* deben aparecer en el mismo orden en que fueron declarados. Si el orden es invertido, el XML es considerado inválido.

El elemento *sequence* define lo que es conocido como *modelo de elementos agrupados*. Además de *sequence*, podemos usar también *choice* y *all* para definir modelos de elementos agrupados. *Choice* indica que sólo uno de los elementos puede aparecer en el documento XML. *All* indica que ninguna o una instancia de cada elemento puede aparecer en el documento XML en cualquier orden.

Declarando atributos

Los atributos son declarados como parte de la definición de un tipo complejo.

```
<complexType name="exampleType">
      <sequence>
            <element name="elem1" type="string"/>
            <element name="elem2" type="string"/>
      </sequence>
      <attribute name="quant" type="int" use="required"/>
      <attribute name="size" type="string" use="optional"/>
</complexType>
```

Por cada atributo, agregamos un elemento *attribute* y le especificamos el nombre, el tipo de dato y su uso.

Contenido simple

Recordemos que un tipo es complejo si tiene atributos y/o elementos hijos. Si un tipo complejo contiene elementos hijos, se dice que tiene contenido complejo. Sin embargo, si un tipo complejo tiene atributos pero contiene solo texto, se dice que es un

tipo complejo con contenido simple. Por ejemplo, el siguiente elemento es un tipo complejo con contenido simple:

```
<example size="Large">test</example>
```

La siguiente es la definición del tipo correspondiente al elemento anterior:

```
<complexType name="exampleType">
      <simpleContent>
            <extension base="stirng">
                  <attribute name="size" type="string" use="required"/>
            </extension>
      </simpleContent>
</complexType>
```

Contenido mixto

Un elemento XML se dice que tiene contenido mixto si contiene tanto texto como elementos hijos. Por ejemplo:

```
<examples>
      Esto es contenido simple
      <elem1>texto</elem1>
      <elem2>texto</elem2>
</examples>
```

La representación de la definición del tipo complejo con contenido mixto en XSD sería:

```
<complexType name="exampleType" mixed="true">
      <sequence>
            <element name="elem1" type="string"/>
            <element name="elem2" type="string"/>
      </sequence>
</complexType>
```

En general, los modelos con contenido mixto no deberían ser usados en aplicaciones de procesamiento de datos.

Tipos anónimos

No siempre es necesario dar nombre a los tipos. Si incluimos la definición del tipo como parte de la declaración de un elemento, no necesitamos dar nombre al tipo.

Los tipos complejos también pueden ser definidos anónimamente de la siguiente manera:

```
<element name="example">
      <complexType>
            <sequence>
                  <element name="elem1" type="string"/>
                  <element name="elem2" type="string"/>
            </sequence>
      </complexType>
</element>
```

Tipos XSD y espacios de nombres

Como hemos visto, cada tipo que definimos tiene un nombre a menos que sea un tipo anónimo. Cuando los desarrolladores crean esquemas con definiciones de tipos, es común que diferentes tipos usen el mismo nombre. Esto puede llevar a colisiones de nombres. Este problema fue resuelto como se hizo con los nombres de los elementos y los atributos en XML. Los tipos pueden pertenecer a un espacio de nombres. [35]

Todos los tipos integrados pertenecen al espacio de nombres de XSD http://www.w3.org/2001/XMLSchema.

Validando con XSD

Además de actuar como una forma de especificar una clase de documento XML, un esquema XSD puede ser usado también para validar documentos XML. Para hacer esto, necesitamos usar un programa de validación que sea capaz de chequear un documento XML contra un esquema y reportar errores de validación. En general, hay dos categorías de programas: Analizadores y editores de XML/XSD. Cuando construimos aplicaciones y diseñamos documentos XML y esquemas, típicamente usamos un editor de esquemas. Cuando una aplicación esta corriendo, debe programáticamente validar documentos XML usando analizadores de validación. [35]

Validando un documento XML usando un esquema

El programa que valida debe, de alguna manera, conocer el esquema a usar para validar el documento. Hay una forma estándar para que un documento XML haga referencia a su esquema: hay dos atributos dependiendo de si usamos o no un espacio de nombres en el documento instancia. Los atributos son llamados *schemaLocation* y *noNamespaceSchemaLocation*. Ambos atributos pertenecen al espacio de nombres *XML Schema Instance* (http://www.w3.org/2001/XMLSchema-Instance). Si los elementos o atributos que estamos validando no pertenecen a ningún espacio de nombres, usamos el atributo *noNamespaceSchemaLocation* para especificar la ubicación física del esquema del documento. El atributo *schemaLocation* nos deja especificar la ubicación física del esquema a usar para validar los elementos y atributos en un espacio de nombres dado. [35]

Por supuesto, querremos validar programáticamente un documento de entrada antes de procesarlo dentro de nuestra aplicación. Para hacer esto usamos un analizador de validación. Las aplicaciones .NET usan la clase *System.Xml.Xml.ValidatingReader* que es un analizador que soporta validación basada en XSD. [35]

Nosotros construiremos Servicios Web que envíen y reciban documentos XML. Por ejemplo, supongamos que tenemos el método *EnviarFactura* que acepta un documento *factura* en un parámetro *string*. Para procesar la factura, el método necesita analizar la factura leyendo los datos del documento y usándolos para generar un objeto *Factura*. Este proceso es conocido como deserializar un objeto desde XML. Similarmente, si un cliente está tratando de invocar al método *EnviarFactura,* debe internamente generar el documento XML a enviar desde un objeto *Factura*. Este proceso es conocido como serializar un objeto a XML.

Si estamos construyendo una aplicación que se basa en la serialización/deserialización de objetos de muchas clases diferentes, la mejor apuesta es incluir serialización en cada clase. Por ejemplo, podemos definir una interfaz llamad *IXmlSerializable* que debe implementar cada clase:

```
public interface IXmlSerializable
{
      string ToXmlString ();
      void FromXmlString (string xmlStr);
}
```

El desarrollador de cada clase debe ser el responsable de implementar los métodos para serializar/deserializar. Para aplicaciones grandes con muchas clases, escribir y mantener el código para serializar/deserializar las clases puede convertirse en una tarea tediosa. Afortunadamente, .NET provee excelente soporte para automatizar la serialización/deserialización de objetos.

Como trabaja la serialización

La librería de clases de .NET incluye un framework para serializar XML bajo el espacio de nombres *System.Xml.Serialization*. La figura siguiente muestra las clases involucradas en el proceso de serialización. Las clases celestes son parte del framework de serialización XML.

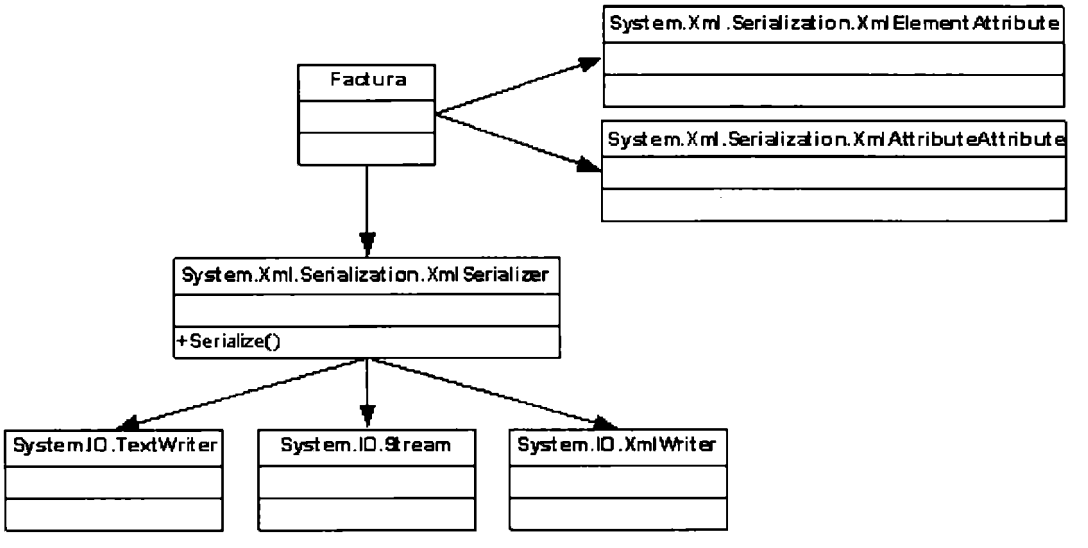


Figura 25. Clases del proceso de serialización de XML.

Asumiendo que tenemos un objeto de la clase *Factura* que queremos serializar, usamos la clase *System.Xml.Serialization.XmlSerializer*. Llamamos al método *Serialize* de esta clase pasando el objeto *Factura* y uno de la siguiente clase: *TextWriter*, *Stream* o *XmlWriter*. *XmlSerializer* usa *reflection* para leer los campos públicos del objeto factura y serializarlo a XML dentro del otro objeto pasado como parámetro. *XmlSerializer* determina automáticamente la estructura del documento XML generado a menos que usemos atributos para influenciar la estructura. La figura anterior muestra dos de los atributos comúnmente usados: *XmlElementAttribute* y *XmlAttributeAttribute*. Estos son atributos del CLR de .NET no atributos XML. Los atributos .NET son clases que heredan de *System.Attribute*. Estas clases son usadas para agregar metadata a los ensamblados que puede ser usada en tiempo de ejecución para agregar funcionalidad. Por ejemplo, agregando *XmlElementAttribute* a un campo público de una clase, le estaremos diciendo al *XmlSerializer* que queremos que este campo sea serializado como un elemento con el nombre especificado. Por ejemplo, para especificar que *invoiceNumber* debe ser serializado como el elemento `<invNo>` debemos escribir:

```
[System.Xml.Serialization.XmlElement("invoNo")]
public string invoiceNumber
```

4.7. Lenguaje de Descripción del Servicio Web

Si otros desarrolladores consumirán un Servicio Web nuestro necesitarán conocer, entre otras cosas, qué métodos Web expone, qué protocolos soporta, la firma de los métodos Web, la ubicación del Servicio Web (URL), etc. Toda esta información y más puede ser expresada en un lenguaje llamado **Web Services Description Language** (WSDL) (Lenguaje de Descripción de Servicios Web).

WSDL es una gramática XML para describir las interfaces, los protocolos y la ubicación de los Servicios Web. La versión 1.1 de la especificación WSDL fue iniciada por IBM Research y Microsoft y puede ser encontrado en http://www.w3c.org/TR/wsdl. Aunque WSDL no es un estándar W3C, ni es requerido para la construcción e invocación de Servicios Web, sin embargo es soportado por muchas herramientas SOAP, como *wsdl.exe*, que lo usan para facilitarle a los clientes la invocación de los Servicios Web [35].

Del servidor, cuando expone un Servicio Web, conseguimos automáticamente un documento WSDL describiendo el servicio. Así, podemos leer el documento para conocer la interfaz del servicio y como invocarlo. En el lado del cliente tendremos dos formas de usar el documento WSDL. Una herramienta de desarrollo, como *wsdl.exe*, puede leer el documento en tiempo de diseño y generar el código cliente necesario para invocarlo. Alternativamente, una herramienta de desarrollo puede leer el documento en tiempo de ejecución para generar los requerimientos necesarios y para procesar sus respuestas [35].

En WSDL, un servicio expone grupos de operaciones, por ejemplo métodos. Cada grupo de operaciones es llamado un *portType* que es aproximadamente análogo a una interfaz en el mundo COM. Para invocar una operación, el cliente envía un mensaje de entrada y regresa un mensaje de salida. El mensaje de entrada contiene los datos de

ida al Servicio Web y el mensaje de salida contiene los datos que regresan desde el servicio. Cada ítem de datos en un mensaje es llamado *part*. El protocolo usado para invocar una operación y el formato de los mensajes de entrada y salida son especificados en un *binding*. El servicio mismo está expuesto al mundo vía uno o más *ports*. Cada port especifica dos cosas: una dirección de red donde está ubicado y el binding a usar con este port. Un servicio puede ser expuesto vía múltiples ports cada uno con un diferente binding [35].

La siguiente figura muestra los componentes de un documento WSDL y como se relacionan unos con otros. Las cajas muestran la relación de contención y las flechas las relaciones de referencia.

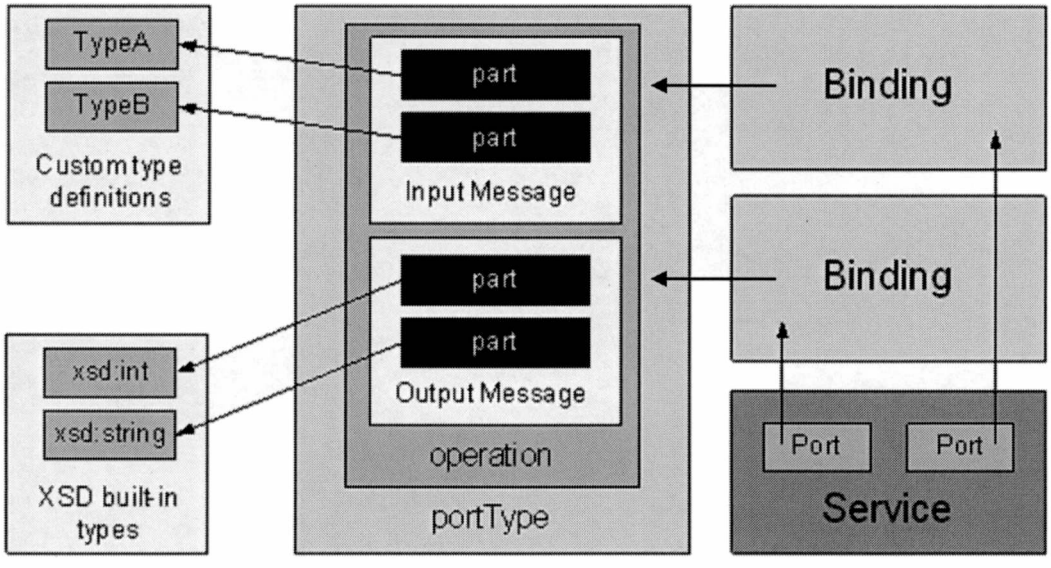


Figura 26. Componentes de un documento WSDL.

Un servicio contiene uno o más ports y cada port referencia un binding. Cada binding referencia un portType, la operación está dentro del portType y cada mensaje que constituye la operación dentro de esta. Cada portType contiene cero o más operaciones. Cada mensaje contiene un mensaje de entrada y otro de salida. Cada mensaje tiene cero o más parts y cada part es de algún tipo de dato. El tipo de la part puede ser un tipo integrado XSD o puede ser un tipo personalizado simple o complejo definido usando XSD [35].

Generar un WSDL para un Servicio Web construido con el Framework .NET consiste simplemente en poner en el navegador la URL del Servicio Web con **?wsdl** al final.

Buscando dentro de éste podemos encontrar elementos **service** que describen el Servicio Web, elementos **operation** que documentan los métodos Web que el servicio soporta, elementos **binding** que documentan los protocolos que los métodos Web soportan y otra información descriptiva.

Para usar el Servicio Web no es necesario leer el WSDL, en lugar de esto se utilizan herramientas que generan una clase **wrapper** conteniendo todos los elementos necesarios para hablar con el Servicio Web.

4.8. Descubrir Servicios Web

Una vez que un cliente tiene un contrato WSDL describiendo un Servicio Web, tiene toda la información necesaria para hacer llamadas al Servicio Web. Pero cuando publicamos un Servicio Web haciendo que esté disponible sobre un servidor Web, los clientes deben encontrar donde está el contrato WSDL, es más, deben conocer la existencia de nuestro Servicio Web.

Las compañías necesitan una forma de anunciar sus Servicios Web y los clientes una forma de descubrirlos [6]. La solución llega en dos partes: *Universal Description, Discovery, and Integration* (UDDI) y DISCO. El primero es un directorio de Servicios Web globales que es implementado en sí mismo como un Servicio Web. El segundo es un mecanismo basado en archivos para el descubrimiento de Servicios Web locales, esto es un listado de Servicios Web desde archivos DISCO instalados sobre servidores Web.

UDDI

UDDI provee un servicio central de directorio para publicar información técnica sobre Servicios Web. Es el resultado de una iniciativa de la industria respaldada por un número significante de compañías tecnológicas, incluyendo Microsoft, IBM y Ariba.

UDDI es otro ejemplo sin precedencia a nivel de cooperación industrial alrededor de la adopción de Servicios Web. Muchas compañías consideran que el servicio de directorio para publicar Servicios Web será crucial para que los Servicios Web sean utilizados masivamente. [35]

Como resultado, la especificación UDDI y la infraestructura requerida para soportarlo fue desarrollada cooperativamente por un número de compañías. Una vez que UDDI alcanzó un "razonable nivel de madurez", los miembros del proyecto se comprometieron a enviar la especificación UDDI a un organismo de estándares. [35]

Arquitectura

La infraestructura que soporta UDDI está compuesta por un conjunto de registros y registradores. Un registro contiene una copia completa del directorio UDDI, un registrador provee servicios de registración en representación del cliente. [35]

Un registrador puede ser un IPS, un host de mercado B2B, una compañía individual, o un host de registración en sí mismo. Por ejemplo, Microsoft ofrece un registro y también provee una interfase HTML para crear y mantener registros dentro del directorio. [35]

Como vimos antes, Microsoft e IBM son sólo dos compañías que proveen servicio de registración.

Los registros están basados en el modelo de replicación *single-master*. Un negocio debe elegir un registro en el cual mantener su información. Todas las actualizaciones hechas al directorio serán replicadas a todos los otros registros. Entonces la información actualizada puede ser requerida desde cualquier registro.

API UDDI

Un registro UDDI es en sí mismo un Servicio Web. Este expone una API basada en SOAP para acceder y manipular las entradas dentro del directorio. En vez de mantener datos a través de los registradores, los desarrolladores pueden programar directamente contra la API.

La versión 1 (uno) de la API UDDI expone aproximadamente 30 (treinta) métodos para interactuar con el registro, todos ellos tienen un comportamiento sincrónico. El siguiente ejemplo muestra como esta estructurado un mensaje de requerimiento UDDI:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
      <Body>
            <find_business generic="1" xmlns="urn:uddi-org:api">
                  <name>MyTestBusiness</name>
            </find_business>
      </Body>
</Envelope>
```

Un mensaje UDDI debe cumplir algunos requerimientos mínimos para que sea válido. Algunos de estos elementos requeridos se muestran en negrita en el ejemplo anterior. Ellos son los siguientes:

- La especificación UDDI requiere que el mensaje SOAP esté codificado en UTF-8.
- Los elementos dentro del cuerpo de un documento UDDI deben tener su alcance dentro del espacio de nombres UDDI API. El espacio de nombres UDDI API es identificado por la URI *urn:uddi-org:api*.
- El requerimiento debe contener un *SOAPAction* en el encabezado cuyo valor sea un *string* vacío. Cualquier otro valor será considerado un error.
- La versión de la API apuntada debe estar declarada dentro del cuerpo del mensaje usando el atributo *generic*.

Examinemos el último punto en más detalle. Como cualquier sistema, UDDI continuó evolucionando. Como se hicieron mejoras, la API tuvo que ser modificada para exponer la nueva funcionalidad. La versión 2 de la API ha completado el proceso de revisión, ha sido ratificada por los miembros de UDDi.org y actualmente ya ha sido implementada. La organización UDDI ideó un sistema de versionado para mantener algún grado de compatibilidad hacia atrás.

La versión de la API es referida como su *generic*. El atributo *generic* indica el número de versión de la API apuntada dentro del elemento raíz del cuerpo SOAP.

El registro es responsable de soportar tanto la versión actual como la previa de *generic*.

Los métodos de la API UDDI pueden ser divididos en dos categorías: métodos de búsqueda y métodos de publicación. Los métodos de búsqueda nos permiten buscar y navegar el directorio, y los métodos de publicación nos permiten modificar el contenido del directorio. Los mensajes de los métodos de búsqueda tienen el elemento raíz del *body* prefijado por *find_* o *get_*. Con algunas excepciones, los mensajes para los métodos de publicación tienen el elemento raíz del *body* del mensaje prefijado por *save_* o *delete_*.

La especificación técnica para la API UDDI es publicada como un documento WSDL así como también como un XML Schema. La versión XML Schema se puede ubicar en *http://www.uddi.org/schema/2001/uddi_v1.xsd*. El documento WSDL para la API de búsqueda está ubicado en *http://www.uddi.org/wsdl/inquire_v1.wsdl*, y el documento WSDL para la API de publicación está en *http://www.uddi.org/wsdl/publish_v1.wsdl*. La tabla siguiente describe los métodos de búsqueda.

Mensaje	Descripción
find_binding	Busca por entidades *bindingTemplate* que correspondan a un conjunto específico de criterios
find_business	Busca por entidades *businessEntity* que correspondan a un conjunto específico de criterios
find_service	Busca por entidades *businessService* que correspondan a un conjunto específico de criterios
find_tModel	Busca por entidades *tModel* que correspondan a un conjunto específico de criterios
get_bindingDetail	Obtiene una o más entidades *bindingTemplate* especificas
get_businessDetail	Obtiene una o más entidades *businessEntity* especificas
get_businessDetailExt	Obtiene una o más entidades *businessEntityExt* especificas desde un registro que puede soportar atributos adicionales
get_serviceDetail	Obtiene una o más entidades *businessService* especificas
get_tModelDetail	Obtiene una o más entidades *tModel* especificas

Los métodos *find_* son para búsquedas generales y los métodos *get_* son para obtener información detallada sobre un registro en particular. La tabla siguiente describe los métodos de publicación.

Mensaje	Descripción
delete_binding	Elimina una o más entidades *bindingTemplate* especificas
delete_business	Elimina una o más entidades *businessEntity* especificas
delete_service	Elimina una o más entidades *businessService* especificas
delete_tModel	Elimina una o más entidades *tModel* especificas
discard_authToken	Invalida el token de autenticación previamente obtenido
get_authToken	Obtiene un token de autenticación para el usuario
get_registredInfo	Obtiene una lista de entidades *businessInfo* y *tModelInfo* para un usuario en particular
save_binding	Guarda una o más entidades *bindingTemplate*
save_business	Guarda una o más entidades *businessEntity*
save_service	Guarda una o más entidades *businessService*
save_tModel	Guarda una o más entidades *tModel*

Los métodos de publicación son responsables de manipular los datos dentro del directorio. Las creaciones y las actualizaciones son manejadas por los mensajes *save_*.

Cuando un registro es creado usando un método *save_*, el registro genera un identificador único. Este identificador es pasado a los subsiguientes métodos *save_* llamados para actualizar el registro. Cada mensaje *save_* tiene su correspondiente mensaje *delete_* para eliminación.

Las modificaciones hechas usando los métodos de publicación deben ser hechas de una forma segura. La especificación UDDI declara que todos los mensajes de publicación deben ser intercambiados entre el cliente y el servidor sobre HTTPS. Esto evita que el contenido del mensaje sea modificado durante la transferencia.

Un registro UDDI nos permite modificar y eliminar sólo nuestros registros, por esto debemos incluir un token de autenticación con cada requerimiento para probar nuestra identidad. Este token es pasado como parte de la firma de los métodos *save_* y *delete_*.

Obtenemos un token de autenticación pasando nuestras credenciales al método *get_authToken*.

Microsoft provee un UDDI SDK para desarrollar aplicaciones habilitadas para UDDI. El UDDI SDK incluye un modelo de objetos .NET y basado en COM para simplificar la interacción con el registro.

La mayoría de los desarrolladores no tratan con las API's de UDDI directamente. En lugar de esto usan herramientas de alto nivel como Visual Studio .NET.

DISCO

DISCO, que es una abreviatura de "discovery", es otra tecnología que podemos usar para publicar y descubrir Servicios Web.

El protocolo DISCO fue desarrollado por Microsoft, que actualmente no tiene planes formales de enviar la especificación DISCO a algún cuerpo de estándares. [35] Por lo tanto, ¿por qué DISCO si tenemos UDDI?.

Recordemos que UDDI es un servicio de directorio estructurado administrado centralizadamente. Su capacidad para descubrir Servicios Web esta centrada en la compañía, esto dificulta determinar qué Servicios Web son expuestos por un servidor particular. Para este tipo de consultas necesitamos un mecanismo más descentralizado para localizar Servicios Web.

DISCO permite descubrir qué Servicios Web están corriendo sobre una computadora en particular proveyendo un paradigma orientado a la navegación para localizar un Servicio Web particular. En algunos aspectos, DISCO es similar a la navegación por hipervínculos popularizada por HTML. Podemos hacer público un índice en el nivel superior que contenga referencias a los Servicios Web específicos o a otros archivos DISCO. [35]

Como DISCO soporta un paradigma orientado a la navegación, es apropiado para ambientes de desarrollo. Y como DISCO no requiere registrarse formalmente como UDDI, podemos rápidamente exponer nuestros Servicios Web a otros desarrolladores.

Estos pueden navegar nuestro servidor de desarrollo para descubrir la URL de un Servicio Web particular contra el que necesiten codificar.

DISCO tiene la desventaja que no podemos leer el archivo DISCO si no tenemos su URL.

DISCO y Visual Studio .NET

Por defecto, la aplicación de ayuda para agregar referencias Web de Visual Studio .NET usa archivos DISCO para localizar Servicios Web.

Visual Studio .NET crea automáticamente los archivos DISCO necesarios. Durante la instalación, Visual Studio .NET creará automáticamente un archivo DISCO para el servidor Web por defecto.

El siguiente es un ejemplo de un archivo DISCO creado por un proyecto de Servicio Web:

```
<?xml version="1.0" ?>
<dynamicDiscovery xmlns="urn:schemas-dynamicdiscovery:disco.2000-03-17">
      <exclude path="_vti_cnf">
      <exclude path="_vti_pvt">
      <exclude path="_vti_log">
      <exclude path="_vti_script">
      <exclude path="_vti_txt">
</dynamicDiscovery>
```

El archive DISCO llamado *default.vsdisco* es ubicado en el directorio raíz del servidor Web. Todos los archivos mantenidos sobre Microsoft Internet Information Server (IIS) que contienen la extensión de archivo *.vsdisco* serán manejados por *DiscoveryRequestHandler* dentro de ASP.NET. Sin embargo, *DiscoveryRequestHandler* está deshabilitado por defecto.

Los archivos DISCO que contienen el elemento *dynamicDiscovery* indicará al filtro ISAPI de ASP.NET que busque los directorios que contengan archivos con la extensión *.asmx* y que busque todos los subdirectorios por los archivos con la extensión *.vsdisco*. Cualquier camino listado con un elemento *exclude* será ignorado.

Por ejemplo, supongamos que el archivo de descubrimiento dinámico fue generado sobre un servidor llamado *Development*. El servidor mantendrá un Servicios Web llamado *Invoice* en un subdirectorio con el mismo nombre. En este caso, el archivo DISCO resultante es el siguiente:

```
<?xml version="1.0" encoding="utf-8" ?>
<discovery xmlns="http://schemas.xmlsoap.org/disco/">
      <discoveryRef ref="http://Development/Invoice/Invoice.vsdisco" />
</discovery>
```

El documento DISCO contiene un elemento *discoveryRef* para cada archivo DISCO encontrado dentro de los subdirectorios. Este elemento tiene un propósito similar a un *href* de HTML.

Cuando Visual Studio .NET crea un proyecto de un Servicio Web, crea un archivo DISCO. Este archivo tiene el mismo nombre que el proyecto y la extensión *.vsdisco*. El siguiente es el contenido del archivo *Invoice.vsdisco*:

```
<?xml version="1.0" ?>
<dynamicDiscovery xmlns="urn:schemas-dynamicdiscovery:disco.2000-03-17">
      <exclude path="_vti_cnf">
      <exclude path="_vti_pvt">
      <exclude path="_vti_log">
```

```
        <exclude path="_vti_script">
        <exclude path="_vti_txt">
        <exclude path="Web References">
</dynamicDiscovery>
```

El contenido de este archivo es similar al del archive *default.vsdisco* excepto que éste incluye el directorio *Web Referentes*. Cuando un cliente accede al archivo, el archivo DISCO es generado dinámicamente:

```
<?xml version="1.0" encoding="utf-8" ?>
<discovery xmlns="http://schemas.xmlsoap.org/disco/">
        <contractRef ref="http://Development/Invoice/Invoice.asmx?wsdl"
                     docRef="http://Development/Invoice/Invoice.asmx"
                     xmlns="http://schemas.xmlsoap.org/disco/scl/" />
</discovery>
```

El documento DISCO resultante contiene un único elemento *contractRef* que contiene una referencia al documento WSDL y la documentación HTML del Servicio Web.

Como es costoso explorar directorios cada vez que un archivo DISCO generado dinámicamente es accedido, en lugar de eso podemos optar por exponer un archivo DISCO estático. La forma más sencilla de crear un archivo DISCO estático es reemplazar el archivo por el resultante de la generación dinámica.

4.9. Protocolos de transporte y enlace

Por defecto los Servicios Web soportan tres estilos de enlace al protocolo HTTP: SOAP, HTTP GET y HTTP POST.

Todos los Servicios Web ASP.NET soportan el enlace SOAP. De los tres estilos de enlaces, SOAP es frecuentemente el más usado porque contener datos dentro del mensaje puede ser fuertemente tipado usando XML Schema. Además, los tipos de datos XML pueden ser bastante equivalentes a los tipos de datos .NET. [35]

Sostener enlaces HTTP GET/POST es más limitado que para SOAP. Algunos factores que limitan la habilidad del motor de ejecución de ASP.NET para soportar los enlaces HTTP GET/POST son los siguientes: [35]

- *Encabezados SOAP requeridos:* los enlaces HTTP GET/POST no proveen una forma de enviar y recibir información de encabezado. Si el documento WSDL de un Servicio Web declara que un encabezado debe ser incluido siempre en un mensaje intercambiado entre el cliente y el servidor, el mensaje debe ser codificado usando SOAP. [35]
- *Parámetros de entrada complejos:* ASP.NET no soporta codificación de tipos complejos dentro del par nombre-valor sobre el q*uery string* o en el cuerpo de un requerimiento HTTP. [35]
- *Múltiples parámetros retornados al cliente:* sólo el parámetro de retorno puede ser pasado de vuelta al cliente. ASP.NET no soporta codificación de parámetros de entrada-salida o de salida dentro del mensaje retornado al cliente como resultado de un requerimiento HTTP GET/POST. [35]

Si el Servicio Web expone interfaces relativamente simples, también puede ser expuesto vía HTTP GET y HTTP POST. Estos enlaces son más simples que SOAP.

4.10. Implementación en .NET

Hay muchos medios para escribir Servicios Web. Podemos escribirlos a mano, usando herramientas de Microsoft, IBM y otras compañías o podemos usar el Framework .NET. Escribir Servicios Web con el Framework .NET ofrece dos ventajas sobre todos los otros métodos:

- El Framework .NET hace que escribir Servicios Web sea extremadamente fácil.
- Los Servicios Web escritos con el Framework .NET son aplicaciones manejadas, lo que significa que contamos con todas las ventajas de la plataforma .NET.

Instalar un Servicio Web es tan simple como copiarlo en una carpeta en el servidor Web que sea direccionable a través de una URL.

Algunos principios importantes cuando programamos Servicios Web usando el Framework .NET son:

- Los Servicios Web son implementados en archivos ASMX. ASMX es una extensión de archivo especial registrada por ASP.NET en el archivo Machine.config.
- Los archivos ASMX comienzan con directivas **@WebService**. Como mínimo la directiva debe contener un atributo **class** que identifique la clase que constituye el Servicio Web.
- A las clases de los Servicios Web se les pueden poner opcionalmente atributos.
- Los métodos Web son declarados etiquetando métodos públicos en las clases de los Servicios Web con el atributo **WebMethod**. Además, podemos construir otros métodos de ayuda, que son métodos usados internamente por los métodos Web pero que no son expuestos como métodos Web omitiendo el atributo WebMethod.
- HTTP, XML y SOAP están ocultos, no tenemos que tratar con ellos porque el Framework .NET lo hace por nosotros.

Podemos mover las clases del Servicio Web fuera del archivo ASMX y compilarlas separadamente dentro de una DLL. Esto permite escribir Servicios Web en lenguajes que no son soportados nativamente.

4.11. Clientes de Servicios Web

Los clientes de los Servicios Web son aplicaciones que usan métodos Web. Escribir clientes de Servicios Web es fácil gracias al soporte de alto nivel prestado por

la Librería de Clases del Framework .NET y el generador de código llamado **wsdl.exe**. Si tenemos el documento WSDL o la URL del archivo DISCO de un documento WSDL que describa al Servicio Web, podemos hacer llamadas al Servicio Web inmediatamente.

El concepto a captar cuando escribimos clientes de Servicios Web es el de **proxy** del Servicio Web. Un proxy es un objeto que provee una representación local de un Servicio Web remoto. El proxy será instanciado en el dominio de la aplicación cliente, pero los llamados al proxy llegan, a través de él, al Servicio Web que representa.

Los métodos en la clase proxy son un espejo de los métodos Web del Servicio Web. Por ejemplo, si el Servicio Web expone métodos Web llamados *Sumar* y *Restar*, el proxy también contendrá métodos llamados *Sumar* y *Restar*. Cuando llamamos a uno de estos métodos, el proxy toma los parámetros de entrada e invoca al método Web usando el protocolo encapsulado en él (típicamente SOAP). El proxy nos aísla de los detalles de bajo nivel del Servicio Web y del protocolo que éste usa. Este también procesa el XML que retorna el método Web y lo hace disponible como tipos manejados.

En la mayoría de los casos SOAP es el protocolo indicado. Sin embargo, si los métodos a los que estamos llamando son métodos simples que usan tipos de datos simples, cambiar a Get o Post de HTTP hace los llamados levemente más eficientes reduciendo la cantidad de datos transmitidos.

5. Lotus Organizer

Lotus Organizer es una herramienta que permite al usuario organizar tareas que efectúa habitualmente, tales como agendar citas y reuniones, anotar las actividades a realizar, mantener una lista de contactos y administrar llamadas telefónicas.

Organizer posee una ***Agenda*** que facilita la creación de citas para cualquier momento del día, como así también una lista de ***Recordatorios*** de actividades pendientes y cumplidas; un ***Directorio*** para mantener direcciones y números de teléfonos particulares y de oficina, como así también, la posibilidad de realizar ***Llamadas*** telefónicas, registrando las mismas con su duración y su estado. Además posee un diagrama de ***Planeamiento*** para controlar las actividades o proyectos a largo plazo, una sección ***Notas*** para asentar anotaciones o importar archivos y una sección ***Aniversarios*** para incluir aniversarios u otros acontecimientos. Adicionalmente, mediante el acoplamiento de Lotus Notes, Organizer incorpora el manejo de reuniones dando la posibilidad de especificar el día de una reunión, los invitados a ella y aceptar o postergar una invitación [25].

Como complemento a cada una de estas características, Organizer posee el manejo de Alarmas para activar el aviso de un evento, de Tarifas y Clientes para llevar el control económico del tiempo dedicado a una cita y de Categorías para agrupar cada una de las actividades.

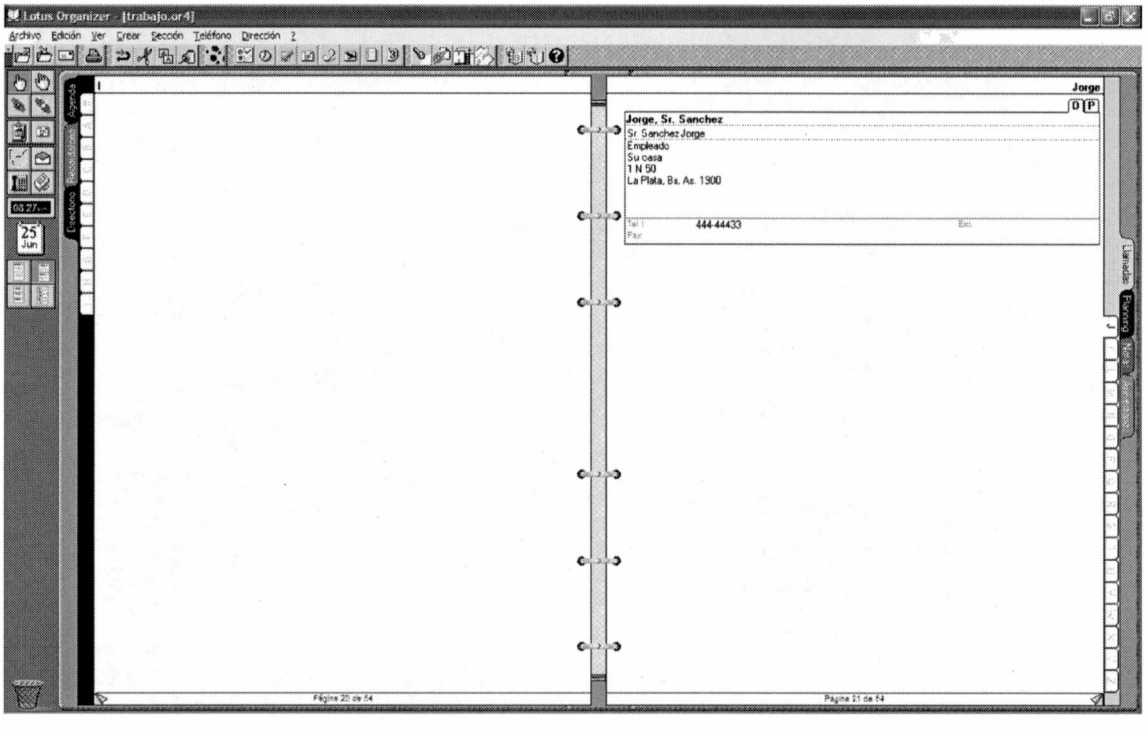


Figura 27. Vista de Lotus Organizer

A continuación se detallan las características, previamente mencionadas, del Lotus Organizer haciendo una división por las secciones que posee.

5.1. Secciones

Agenda

La Agenda permite administrar el calendario y las citas incorporadas dentro de él. Esta sección facilita la realización de las siguientes actividades

- Concertar, planificar, anular y adelantar citas para una fecha y hora deseadas.
- Crear citas periódicas que pueden repetirse con la periodicidad deseada.
- Buscar la siguiente hora disponible para una cita.
- Mostrar informaciones de otras secciones en la sección Agenda; por ejemplo, se puede mostrar la relación Recordatorio de cada día en el día de la Agenda.
- Asignar una Tarifa o una Alarma a una cita.
- Clasificar una cita como confidencial y por categoría.
- Indicar a Organizer que muestre un aviso cuando existan citas coincidentes.
- Establecer vistas diarias, semanales y mensuales para consultarla.
- Trabajar con varios Calendarios simultáneamente, por ejemplo, el propio junto con el de otro usuario.

Recordatorio

El Recordatorio mantiene una lista de asuntos pendientes para así desarrollar toda la planificación semanal o a largo plazo. Esta sección ofrece la posibilidad de:

- Crear tareas.
- Agrupar las tareas por categoría.
- Determinar la prioridad de las tareas.
- Agrupar las tareas según el estado. Dicho estado puede ser: Vencida, Pendiente, Futura o Cumplida.
- Asignar una Tarifa o una Alarma a una tarea.
- Ordenar las tareas por fecha de inicio, fecha límite o fecha de finalización.
- Mostrar las tareas de la sección Recordatorio en la sección Agenda, para poder visualizar todas las tareas en su Agenda.

Directorio

El Directorio contiene las direcciones y teléfonos de personas. Con él se pueden realizar las siguientes tareas:

- Ordenar los datos por apellido, empresa, categoría, código postal o utilizar un campo de dirección personalizado.
- Obtener los datos con la información de todos los campos, el nombre y la dirección, el nombre e información de contacto o el nombre y el número de teléfono.
- Crear fichas de dirección duplicadas para contactos múltiples dentro de la misma empresa, sin tener que copiar o volver a especificar reiteradamente la misma información acerca de la empresa. Esto se utiliza cuando se poseen dos o más contactos en la misma empresa. Para esto, Organizer brinda la opción de duplicar automáticamente toda la información acerca de una empresa cuando se crea la ficha de dirección para todos lo contactos posteriores al primer contacto ingresado para dicha empresa.

Llamadas

La sección Llamadas permite mantener un registro de las llamadas realizadas y recibidas. Posibilitando además:

- Registrar tanto llamadas entrantes como salientes, incluida la información acerca de quién la realizó o a quién iba dirigida y su tiempo de duración.
- Llevar un control del estado de las llamadas, en donde pueden adoptar uno de entre varios estados, tales como, contestada, recibida u ocupada.
- Registrar de forma automática la duración de la llamada.
- Asignar una tarifa a la llamada.
- Planificar llamadas futuras de manera tal de poder asignarle una alarma acústica de modo que avise cuando dicha llamada debe efectuarse.
- Marcar nuevamente aquellos números de las llamadas cuyo estado sea no contestadas.
- Marcar cualquier número que aparezca en la sección Directorio, Llamadas o en cualquier otra sección.
- Reflejar llamadas en la sección Agenda

Planeamiento

El objetivo de la sección Planeamiento no es otro que la preparación y planificación de actividades que vayan a tener lugar durante uno o más días. La sección Planeamiento por lo tanto facilita:

- La asignación de distintas actividades en forma de bloques, como por ejemplo, Conferencias, Reuniones, Vacaciones.
- La visualización de acontecimientos a lo largo de un trimestre o de todo un año.
- La visualización de los acontecimientos en la sección Agenda, de modo que pueda estar al tanto de las mismas a diario.
- La reflexión de actividades de la sección Planeamiento en la sección Agenda, y la reflexión de citas de la sección Agenda en la sección Planeamiento.

La sección Planeamiento es la representación electrónica del plano mural en el que se suelen marcar actividades, tareas y fechas.

La sección Planeamiento proporciona un conjunto de colores que hacen las veces de claves para marcar y distinguir actividades como por ejemplo, Vacaciones, Reuniones o Ferias de muestras. Se puede cambiar el texto descriptivo de estos bloques para que se ajusten a nuestra necesidad.

Se puede crear más de una actividad en un día dado a fin de dar cabida a varias actividades en un solo día. La duración de una actividad se puede especificar con una fecha de finalización, de manera que Organizer ajuste automáticamente el número de días para la actividad, o con un número concreto de días, de manera que Organizer ajuste automáticamente la fecha de finalización.

Notas

Las Notas representan un anotador en el cual se puede agregar texto, modificándolo con los distintos formatos de fuentes tal cual lo hace un procesador de texto. Además brinda la posibilidad de insertar mapas de bits y gráficos, como así también crear objetos OLE incrustados y vinculados. Como complemento a esto puede manejar otro tipo de información, tal como: listas, memorandos, hojas de cálculo, gráficos de ventas, diagramas, organigramas, mapas y logotipos.

Aniversarios

La utilidad Aniversarios facilita el registro de las fechas importantes a recordar en el año, tales como cumpleaños, feriados, etc. Además brinda facilidades para:

- Especificar que un aniversario se repita automáticamente cada año.
- Ordenar y visualizar los aniversarios por año, mes, categoría o signo de zodíaco.
- Asociar una alarma para un aniversario de forma tal de poder recordarlo cuando llegue su fecha.
- Indicar que los aniversarios aparezcan automáticamente en la sección Agenda para que puedan verse en el contexto del trabajo diario.

5.2. Lotus Organizer API

La API (Application Programming Interface) es un conjunto de funciones que permiten interactuar con los archivos de datos de Lotus Organizer. Un desarrollador puede usar esta API para consultar o cambiar registros dentro de un archivo de Lotus Organizer. La mayoría de las operaciones descriptas anteriormente para cada una de las

secciones, pueden ser programadas mediante invocaciones a funciones provistas por la API [29].

Como características de esta API, podemos encontrar:

- Facilita el rápido desarrollo de aplicaciones usando una simple interface de llamadas a funciones de API.
- Permite a las aplicaciones comunicarse e integrarse con Lotus Organizer sin interacción por parte del usuario.
- Extiende las características de Lotus Organizer usando su infraestructura existente.

La API de Organizer incluye un gran conjunto de funciones distribuidas en los siguientes grupos [29]:

- Alarm. Procesa registros de alarma.
- Field. Trabaja sobre uno o más campos de las secciones de Organizer.
- File. Trabaja sobre un archivo de Organizer.
- Initialization. Describe cómo inicializar y terminar una sesión de la API.
- Record. Trabaja sobre uno o más registros de Organizer.
  - Finding. Localiza registros en las secciones de Organizer.
  - Group Scheduling. Identifica información de programación de reuniones en grupo.
  - Inserting. Crea registros en las secciones de Organizer.
  - Linking. Relaciona registros.
  - Repeating. Crea registros repetitivos.
- Section. Trabaja sobre una o más secciones de Organizer.

Los nombres de las funciones de la API de Organizer constan de cuatro partes:

- OrgAPI. Prefijo que usa la API de Organizer para identificar fácilmente sus funciones.
- Grupo. El nombre del grupo de la función.
- Verbo. Indica qué hace la función.
- Modificador. Indica sobre qué actúa la función (no es usado siempre).

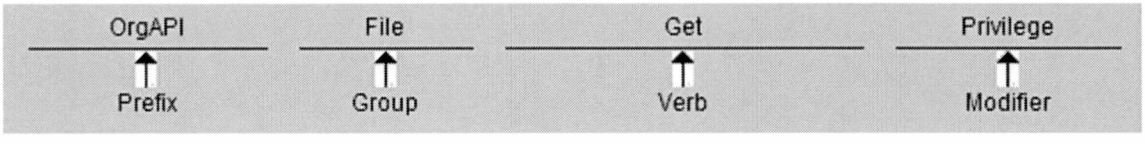


Figura 28. Ejemplo de partes del nombre de una función (*OrgAPIFileGetPrivilege*)

Por otra parte, la API también maneja un conjunto de tipos de datos, algunos comunes, otros para insertar registros y otros específicos de las secciones.

Esta API es utilizada dentro de un componente COM+ para acceder a la información de la herramienta de gestión de tiempos.

6. XML

6.1. Introducción

En unos pocos años, el Extensible Markup Language (XML) ha crecido. Mientras que HTML está diseñado para expresar apariencia, XML esta diseñado para expresar información en crudo alejado de cualquier notación implícita de cómo los datos deben ser mostrados. Este es un lenguaje simple que está basado enteramente en texto, haciendo esto que se ubique particularmente bien para viajar sobre protocolos basados en texto como HTTP, y no tiene tags predefinidos como los tiene HTML. XML provee las reglas, nosotros proveemos el resto.

XML provee un formato de datos común para compañías que quieren intercambiar documentos. Este es usado por los Servicios Web para codificar mensajes y datos de una manera independiente de la plataforma. Este es usado incluso para construir sitios Web, donde sirve como una herramienta para separar claramente el contenido de la apariencia.

Lo que hace al XML importante es el hecho que la industria de la computación lo ha aceptado como un estándar y que existen numerosas herramientas para leer y escribir XML. Si alguien tiene un gran archivo de texto formateado libremente conteniendo miles de registros y nuestro trabajo es llevar los registros dentro de una base de datos, probablemente terminemos escribiendo un analizador para extraer los registros del archivo y escribirlos en la base de datos. Si el archivo de texto es un archivo XML, nuestro trabajo es mucho más simple. Podemos usar uno de los muchos analizadores de XML ya disponibles la leer los registros. XML no nos hace la vida más fácil por ser un gran lenguaje, nos hace la vida más fácil porque herramientas para leer, escribir y manipular datos XML son casi tan comunes como los procesadores de texto.

6.2. XML Elemental

XML es un lenguaje para describir datos y su estructura. Los datos XML están contenidos en un documento, que puede ser un archivo, un stream o cualquier otro medio de almacenamiento, real o virtual, que sea capaz de mantener texto. Un documento XML correcto comienza con la siguiente declaración XML, que identifica al documento como un documento XML y especifica la versión del XML contenido en el documento:

```
<?xml version="1.0"?>
```

La declaración puede también incluir un atributo *encoding* que identifica el tipo de caracteres contenidos en el documento. Por ejemplo, la siguiente declaración especifica que el documento contiene caracteres del conjunto de caracteres Latin-1 para Windows 95, 98 y ME:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

El atributo *encoding* es opcional si el documento consiste de caracteres UTF-8 o UTF-16 porque un analizador XML puede inferir la codificación de los primeros cinco caracteres del documento: "`<?xml`". Los documentos que usen otras codificaciones deben identificarlas para asegurarse que los analizadores de XML podrán leerlos.

La declaración XML es seguida del elemento raíz del documento, al cual usualmente nos referimos como el *document element*. En el siguiente ejemplo, el document element es llamado *Guitars*:

```
<?xml version="1.0"?>
<Guitars>
    ...
</Guitars>
```

El document element no es opcional, todo documento debe tener uno.

Los nombres de los elementos pueden consistir de letras o guiones bajos seguidos por letras, dígitos, puntos, guiones y guiones bajos. Los espacios no están permitidos en los nombres de los elementos.

6.3. Elementos

Los elementos son los bloques de construcción de los documentos XML. Los elementos pueden contener datos, otros elementos, o ambos, y están siempre delimitados por tags de inicio y finalización. XML no tiene elementos predefinidos, los elementos se definen de acuerdo a la necesidad de describir adecuadamente los datos contenidos en un documento XML [24]. El siguiente documento describe una colección de guitarras:

```
<?xml version="1.0"?>
<Guitars>
      <Guitar>
            <Make>Gibson</Make>
            <Model>SG</Model>
            <Year>1997</Year>
            <Color>Tobacco Sunburst</Color>
            <Neck>Rosewood</Neck>
      </Guitar>
      <Guitar>
            <Make>Fender</Make>
            <Model>Stratocaster</Model>
            <Year></Year>
            <Color>Black</Color>
            <Neck>Maple</Neck>
      </Guitar>
</Guitars>
```

En este ejemplo, *Guitars* es el document element, los elementos *Guitar* son hijos de *Guitars*, y *Make*, *Model*, *Year*, *Color* y *Neck* son hijos de *Guitar*. Los elementos *Guitar* no contienen datos, solo otros elementos, pero *Make*, *Model*, *Year*, *Color* y *Neck* contienen datos. La línea `<Year></Year>` significa un elemento vacío, uno que no contiene datos ni a otros elementos. Los elementos vacíos son perfectamente válidos en XML. Un elemento *Year* vacío puede opcionalmente ser escrito de la siguiente manera:

```
<Year/>
```

A diferencia de HTML, XML requiere que los tag de inicio sean acompañados con tags de cierre. Por lo tanto, el siguiente XML no es válido:

```
<Year>1977
```

También a diferencia de HTML, XML es case-sensitive. Un elemento `<Year>` cerrado por un tag `</year>` no es válido porque la apertura del tag comienza con mayúscula y el cierre con minúscula.

6.4. Atributos

XML nos permite adjuntar información adicional a los elementos incluyendo atributos en el tag de inicio de los elementos [24]. Los atributos son pares nombre/valor. El siguiente elemento *Guitar* expresa el año como atributo en vez de como elemento hijo:

```
<Guitar Year="1977">
      <Make>Fender</Make>
      <Model>Stratocaster</Model>
      <Color>Black</Color>
      <Neck>Maple</Neck>
</Guitar>
```

El valor de los atributos debe ser encerrado entre comillas simples (') o dobles (") y puede incluir espacios. El valor de un atributo delimitado por comillas simples puede contener comillas dobles y viceversa. Los nombres de los atributos están sujetos a las mismas restricciones que los nombres de los elementos y por lo tanto no pueden incluir espacios. El número de atributos en un elemento puede decirse que no tiene límites.

Cuando definimos la estructura de un documento, a veces no esta claro si un ítem debe ser un atributo o un elemento. En general, los atributos deben ser usados para definir datos fuera de la agrupación que significa el elemento y los elementos definen datos que son íntegros para el documento.

6.5. CDATA, PCDATA y Entidades referenciadas

El contenido de datos textual de un elemento XML puede ser expresado como caracteres de datos (CDATA), caracteres de datos analizados gramaticalmente (PCDATA) o combinaciones de los dos. Los datos que aparecen entre tags *<![CDATA[* y *]]>* son CDATA, cualquier otro dato es PCDATA. El elemento siguiente contiene PCDATA: [24]

```
<Color>Tabacco Sunburst</Color>
```

El elemento siguiente contiene CDATA:

```
<Color><![CDATA[Tobacco Sunburst]]></Color>
```

Y este otro contiene ambos: [24]

```
<Color>Tobacco <![CDATA[Sunburst]]></Color>
```

Los analizadores de XML ignoran los CDATA pero analizan los PCDATA, esto es los interpreta como parte del lenguaje. La implicación práctica es que podemos poner cualquier cosa entre los tags *<![CDATA[* y *]]>* que para el analizador de XML no tendrá importancia. Los datos que nos estén encerrados entre estos tags deben cumplir con las reglas del XML. [24]

El porqué el XML distingue entre CDATA y PCDATA es que ciertos caracteres, como <, > y &, tienen un significado especial en XML y deben ser encerrados dentro de secciones CDATA si deben ser usados literalmente.

Otra forma de incluir estos caracteres dentro de un documento XML es reemplazarlos con *entity referentes* (entidades referenciadas). Una entidad referenciada es un string encerrado entre los símbolos *&* y *;*. XML predefine las siguientes entidades: [24]

Símbolo	Entidad correspondiente
<	lt
>	gt
&	amp
'	apos
"	quot

También podemos representar caracteres en PCDATA con *carácter referentes* (caracteres referenciados), que no son otra cosa que códigos de caracteres numéricos encerrados entre los símbolos *&#* y *;*.

Los caracteres referenciados son útiles para representar caracteres que no pueden escribirse desde e teclado. Las entidades referenciadas son útiles para escapear ocasionalmente caracteres especiales, pero para grandes cantidades de texto con contenido arbitrario, las secciones CDATA son más convenientes. [24]

6.6. Espacio de nombres

Los espacios de nombres, que están documentados en la especificación de espacios de nombres de XML *http://www.w3.org/TR/REC-xml-names*, son un componente crucial de XML. Estos son un mecanismo para calificar el nombre de elementos y atributos para evitar colisiones de nombres. El ejemplo siguiente define tres espacios de nombres y tres prefijos de los espacios de nombres. Usa los prefijos de los espacios de nombres para calificar sus elementos para que los elementos no choquen si usamos en el mismo documento otros elementos que tienen el mismo nombre pero diferentes definiciones: [45]

```
<?xml version="1.0"?>
<win:Guitars>
      xmlns:win="http://www.wintellect.com/classic-guitars"
      xmlns:gibson="http://www.gibson.com/finishes"
      xmlns:fender="http://www.fender.com/finishes"
      <win:Guitar>
            <win:Make>Gibson</win:Make>
```

```
            <win:Model>SG</win:Model>
            <win:Year>1977</win:Year>
            <gibson:Color>Tobacco Sunburst</gibson:Color>
            <win:Neck>Rosewood</win:Neck>
      </win:Guitar>
      <win:Guitar>
            <win:Make>Fender</win:Make>
            <win:Model>Stratocaster</win:Model>
            <win:Year>1990</win:Year>
            <fender:Color>Black</fender:Color>
            <win:Neck>Maple</win:Neck>
      </win:Guitar>
</win:Guitars>
```

XML también soporta el concepto de *espacios de nombres por defecto*. Un espacio de nombres por defecto es declarado con un atributo *xmlns* pero sin prefijo. El elemento en el que el espacio de nombres está declarado y todos sus hijos automáticamente pertenecen al espacio de nombres a menos que se especifique lo contrario.

Los elementos XML usados por una persona o aplicación típicamente no necesitan de espacios de nombres porque se pueden prevenir las colisiones de nombres. Los electos XML para consumo público deben ser calificados con espacios de nombres porque no se tiene el control de como serán usados o que otros elementos serán usados con ellos. [45]

6.7. Validez de documentos y esquemas

Un documento XML que cumple con las reglas de XML se dice que es un *documento bien formado*. [45]

Un *documento válido* es uno que está bien formado y que cumple con el esquema. Los esquemas definen estructuras de documentos aceptables y contenido. [45]

En los primeros días de XML, los desarrolladores usaban *Document Type Definitions* (DTD's) para validar el contenido de documentos XML. Hoy en día ellos usan *XML Schema Definitions* (XSD's), que son descriptos en *http://www.w3.org/TR/xmlschema-1* y *http://www.w3.org/TR/xmlschema-2*. XSD es un lenguaje basado en XML para describir documentos XML y los tipos que ellos contienen. El esquema siguiente define el formato válido de un documento XML describiendo guitarras: [45]

```
<?xml version="1.0"?>
<xsd:schema id="Guitars" xmlns=""
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="Guitars">
      <xsd:complexType>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element name="Guitar">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="Make" type="xsd:string">
                <xsd:element name="Model" type="xsd:string">
                <xsd:element name="Year" type="xsd:gYear" minOccurs="0">
                <xsd:element name="Color" type="xsd:string" minOccurs="0">
                <xsd:element name="Neck" type="xsd:string" minOccurs="0">
```

```
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Este esquema, que debe estar almacenado en un archive XSD, puede ser usado para validar el documento XML que usamos en ejemplos anteriores. Este dice que un documento válido debe contener un elemento *Guitars*, que el elemento *Guitars* debe contener uno o más elementos *Guitar*, y que el elemento *Guitar* debe contener exactamente un elemento *Make* y un elemento *Model* y puede contener un elemento *Year*, un elemento *Color* y un elemento *Neck*, en este orden. Este también dice que los elementos *Make*, *Model*, *Color* y *Neck* contienen datos string, mientras *Year* contiene un año calendario Gregoriano. [45]

Los esquemas están para ser consumidos por computadoras, no humanos. Numerosas herramientas de software están disponibles para validar documentos XML contra esquemas y para generar esquemas desde documentos XML. Una de estas herramientas es la utilidad Xsd.exe que viene con el SDK del .NET Framework.

6.8. Analizadores XML

La mayoría de los analizadores XML usa una de dos API's populares: DOM o SAX. DOM está puesto para *Document Object Model* y está descrito en *http://www.w3.org/TR/DOM-Level-2-Core*. SAX está puesto para *Simple API for XML* y es un estándar no oficial, esto es no W3C, en la comunidad Java y está documentado en *http://www.saxproject.org*. Ambas API's definen una interfaz programática que abstrae la naturaleza física de los documentos XML, pero difieren en la forma que trabajan sobre este. [45]

SAX es una API basada en eventos. Proveemos un analizador SAX con una o más interfaces que contienen conjuntos conocidos de métodos de comunicación de retorno. Consideremos el siguiente documento XML: [45]

```
<Greeting>Hello, World</Greeting>
```

Una aplicación que quiere leer este documento usando un analizador SAX implementa una interfaz conocida conteniendo métodos llamados *startDocument*, *endDocument*, *startElement*, *endElement* y *carácter* junta a otros. De acuerdo a como el analizador se mueve a través del documento, este invoca a estos métodos sobre el cliente en el siguiente orden: [45]

```
startDocument      // Señala el comienzo del documento
startElement       // Señala el comienzo del elemento Greeting
characters         // Transmite "Hello, World"
...
endElement         // Señala el final del elemento Greeting
endDocument        // Señala el final del documento
```

Las llamadas a *startElement* y *endElement* son acompañados por los nombres de los elementos. Los puntos suspensivos que siguen a *character* indican que este es llamado una cantidad indeterminada de veces. Algunos analizadores SAX pueden

llamar a este una vez y pasar "Hello, world" en un chunk, pero otros pueden llamarlo varias veces y transmitirlo en partes. SAX es extremadamente útil para analizar grandes documentos porque no requiere que el documento entero sea leído en memoria de una vez. La principal desventaja de SAX es que este es *forward-only*, una API basada en streams, no podemos movernos arbitrariamente hacia delante y hacia atrás dentro del documento. Ni tampoco podemos identificar fácilmente relaciones entre ítems en el documento.

DOM es una API alternativa que lee el documento en memoria y soporta acceso aleatorio a los ítems contenidos en el documento. Microsoft provee un analizador al estilo DOM en la DLL llamada MSXML.dll, mejor conocida como *MSXML* simplemente. Las nuevas versiones de MSXML soportan SAX también. Ítems individuales dentro de un documento (elementos, atributos, comentarios, textos, etc.) están representados como *nodos*. [45]

Todos los nodos de un documento serán representados en forma de árbol. Cada nodo es un objeto que provee métodos y propiedades para navegar el árbol y extraer el contenido.

7. Emulador de teléfono IP

Para el desarrollo del proyecto se utilizó un emulador de teléfono IP de Cisco Systems, llamado Cisco Phone Services Emulator 1.0 (PSE), provisto por la misma compañía.

El emulador muestra una pantalla y botones dando el aspecto de un teléfono IP real, permitiendo las mismas funcionalidades que éste presta. Además contiene un campo de texto en donde se puede ingresar una URL para conectarse con un servidor web.

Con el PSE existe la posibilidad, tal como se presenta en un browser genérico, de visualizar u ocultar las secciones de Favoritos y Debug. Además, se puede ir a un paso atrás usando el botón Back, parar la navegación con el botón Stop, refrescar la pantalla con el Refresh o retornar a la página de inicio usando Home.

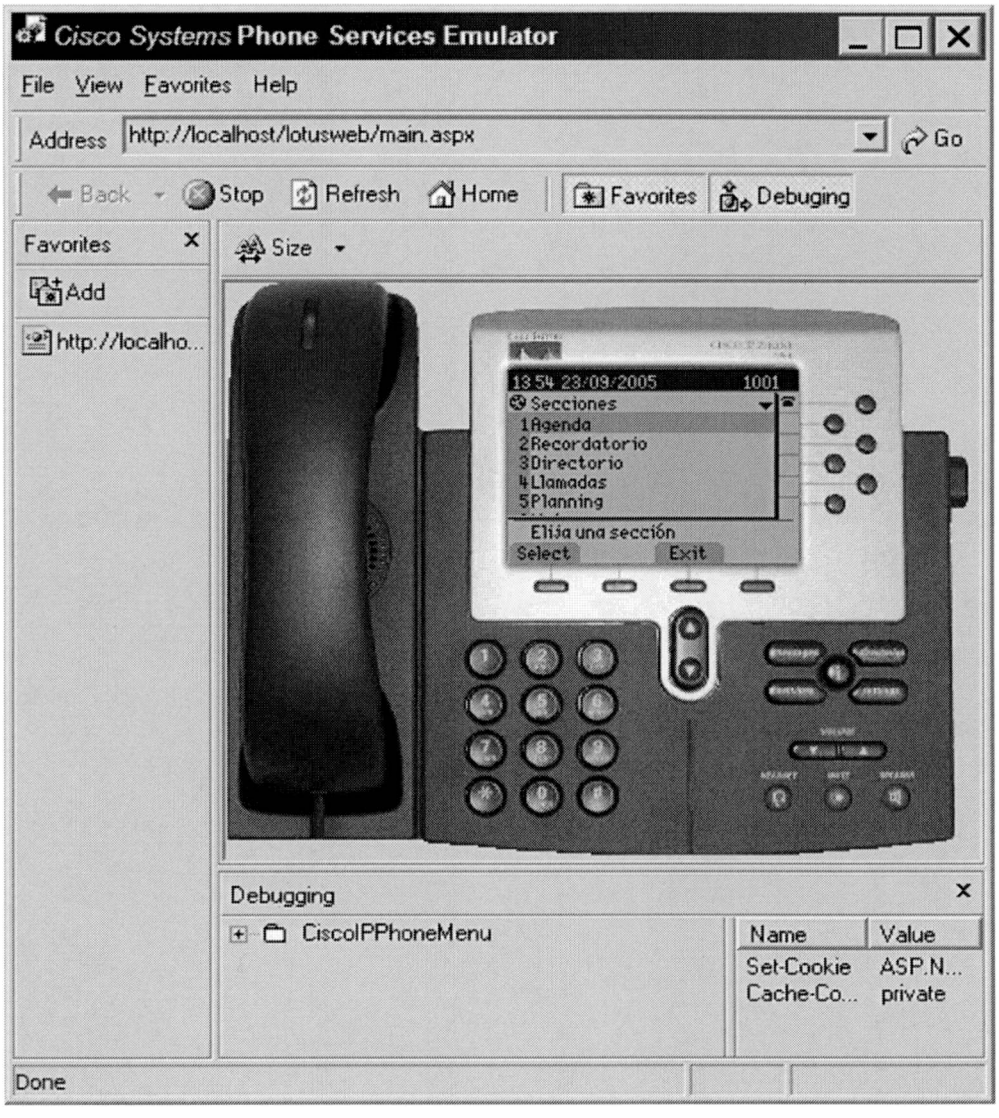


Figura 29. Phone Service Emulator

De esta manera, el emulador tiene la posibilidad de actuar como cliente de una aplicación Web haciendo requerimientos HTTP. Las respuestas enviadas por parte de esta aplicación Web deben ser páginas conteniendo XMLs interpretables por un teléfono IP. El emulador del teléfono interpreta el XML y muestra el contenido en su display.

Una vez mostrada la pantalla en el emulador, se puede navegar a través de la misma usando los botones del emulador, tal como se haría con un teléfono IP real.

Este emulador fue una herramienta fundamental en el desarrollo de la aplicación integradora, ya que la generación de los XMLs y la navegación de la aplicación Web se pudieron testear usando el display y los botones, obteniendo el mismo comportamiento que se esperaba de un teléfono real.

8. Desarrollo de la aplicación integradora

El objetivo del desarrollo de la aplicación fue poder manejar la información de la herramienta de gestión de tiempos a través de un teléfono IP ubicado en un sitio local o remoto a dicha herramienta. De esta manera, un usuario que maneje una aplicación de gestión de tiempos en la red corporativa de su empresa, pueda hacer lo propio desde cualquier ubicación geográfica en donde haya un teléfono IP con conexión a Internet.

La definición de la arquitectura fue un punto fundamental para lograr la integración entre estos dos elementos. Este principal objetivo de **integración** fue lo que guió las distintas etapas del desarrollo y lo que se buscó, en definitiva, como meta a lograr.

Por un lado contábamos con un emulador de teléfonos IP, el cual provee la misma funcionalidad que un teléfono IP para usar en el desarrollo de aplicaciones. Mientras que por otro lado, teníamos la aplicación de gestión de tiempos, en particular Lotus Organizer.

En un principio, tuvimos que definir la forma de interacción con cada una de las partes por separado. Así, establecimos que la manera más apropiada de interactuar con un teléfono IP es mediante XMLs con un formato específico e interpretable por estos dispositivos. Por otra parte, después de buscar varias posibilidades, optamos por la utilización de una API como forma de obtener y enviar información desde y hacia la aplicación de gestión de tiempos, principalmente porque dicha API expone todas las funcionalidades requeridas y es provista por el mismo fabricante de la herramienta [29].

Una vez encontrada la forma de interacción con cada uno por separado, no topamos con la necesidad de integración entre estos dos elementos, para lo cual tuvimos que hallar algo que sea capaz de generar e interpretar los XMLs propios de los teléfonos IP y que, además, los transforme de manera tal de poder interactuar con la aplicación de gestión de tiempos a través de la API. Luego de estudiar varias alternativas posibles, descubrimos que la arquitectura más aconsejable fue implementar un servicio que exponga las funcionalidades de la aplicación de gestión de tiempos basándose en estándares, es decir, que sea capaz de utilizar protocolos de transporte comunes, que permita intercambiar mensajes XML, de acuerdo a un conjunto sencillo de reglas (tal como SOAP) y que permita ser expuesto no sólo en la red IP local sino también a través de Internet y, de esta manera, permitir que se utilice el teléfono IP para acceder a los datos de la aplicación de gestión de tiempos desde Internet. Así arribamos a la conclusión que convenía construir un Servicio Web para lograrlo, junto con el aditamento de una aplicación Web que interactuara con ese Servicio Web y que fuera capaz de comunicarse, a través de HTTP, con el teléfono IP, enviando y recibiendo los XMLs formateados de manera tal que sean interpretables por el teléfono y así poder manejar, a través de su display, la información de la aplicación de gestión de tiempos. Además, la elección de desarrollar un Servicio Web lo más genérico posible que exponga las funcionalidades de la herramienta de gestión de tiempos, contaba con la ventaja de hacer extensible su uso por una aplicación Web particular o Servicio WAP que se encargue de mostrar la información de dicha herramienta de manera similar a la que lo haría el teléfono IP.

Por último, restaba determinar sobré qué plataforma implementar esta arquitectura, para lo cual tuvimos que estudiar cuáles de ellas podrían encuadrar en este esquema, eligiendo finalmente la plataforma .NET de Microsoft, ya que es una nueva generación de tecnología basada en Servicios Web, que permite el uso de estándares, posee un lenguaje orientado a objetos como C# [33] y tiene compatibilidad con la API que nos permite acceder a la información de la aplicación de gestión de tiempos.

Así, con todo esto, se buscó tratar de satisfacer los siguientes objetivos:

- Permitir la publicación de funciones de la herramienta de gestión de tiempos tanto en una red IP local como en Internet, de esta manera, un usuario de la herramienta puede acceder a su información desde cualquier situación geográfica en la cual haya un teléfono IP.
- Posibilitar la consulta, creación y modificación de los datos del usuario de manera amigable utilizando las funcionalidades que brinda un teléfono IP.
- Utilizar tecnologías innovadoras tales como la telefonía IP y los Servicios Web.
- Utilizar estándares masivamente difundidos como lo son HTTP, SOAP y XML.

La utilización de los estándares tuvo un papel importante en este desarrollo dado que se tuvieron que integrar distintas tecnologías y diferentes fabricantes como los son los teléfonos IP de Cisco Systems, la herramienta Lotus Organizer de IBM y la plataforma .NET de Microsoft Corporation.

8.1. Diseño

Acompañando a la decisión sobre el uso de las tecnologías mencionadas, se encontraba la elección del diseño a seguir. Así, se optó por tomar un diseño basado en un modelo multi-capa, en donde participan:

- La herramienta de gestión de tiempos, como almacenadora de la información,
- El Servicio Web, el cual utiliza un componente COM encargado de la interacción con dicha herramienta,
- La aplicación Web, la cual se comunica con el Servicio Web y finalmente interactúa con el teléfono IP
- El teléfono IP, actuando como cliente final.

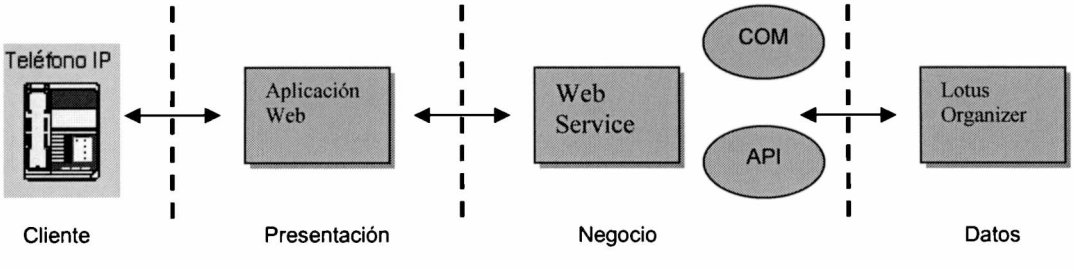


Figura 30. Capas y elementos.

La capa de Datos es la que almacena y mantiene la información de la herramienta de Gestión de tiempos, es decir, las citas de la Agenda, la lista de Recordatorios, los contactos del Directorio, etc.

La capa de Negocio, instanciada en el Servicio Web, es la que se encarga, por un lado, de consultar la información de la capa de Datos y convertirla en XML para que pueda ser interpretada con un formato estándar por la capa de presentación. Por otro lado, recibe requerimientos desde la capa de presentación para consultar y modificar los datos almacenados, procesando los mismos para poder interactuar con la capa de datos. Estas operaciones de interacción con la capa de datos la realiza mediante un componente COM+ que utiliza la API para manipular la información.

La capa de Presentación, personalizada en la aplicación Web, actúa ante un requerimiento del cliente, solicitando servicios a la lógica de negocio para poder acceder a la información. Una vez que recibe los datos, los convierte a un XML interpretable por el teléfono IP y luego los envía al cliente.

El cliente, en este caso un teléfono IP, se comunica con la capa de presentación y, así, permite el manejo de los datos utilizando la pantalla y el teclado.

La motivación por lo cual se realizó dicha elección fue para separar la parte cliente de la lógica de negocio por un lado, y para implementar componentes que puedan ser reutilizados para propósitos similares. Así, por ejemplo, tal como fuera mencionado anteriormente, se buscó desarrollar un Servicio Web utilizando estándares, de manera tal, que luego pueda ser utilizado por otra aplicación Web, una aplicación Win32, un servicio WAP, etc.

8.2. Componentes

En conclusión, dentro del contexto expuesto anteriormente y, desde el punto de vista del desarrollo de la aplicación integradora, se distinguieron varios participantes en la implementación:

- La *aplicación de gestión de tiempos*, en nuestro caso el Lotus Organizer 4.1, el cual fue descripto anteriormente en la sección *Lotus Organizer*.
- Un *Servicio Web*, encargado de obtener y enviar la información desde y hacia la herramienta de gestión de tiempos, utilizando para ello un componente COM+ que usa la Lotus Organizer 4.1 API.
- Una *aplicación Web*, encargada de enviar, usando el protocolo HTTP, los XML interpretables por el teléfono IP, formateados de manera tal que la información que ellos contienen pueda ser mostrada en el display del teléfono. Además esta aplicación Web también está encargada de recibir e interpretar las respuestas del teléfono.
- El *teléfono IP*, utilizado como dispositivo para manejar la información de la herramienta de gestión de tiempos. En el desarrollo se utilizaron como modelo los teléfonos IP de Cisco Systems, usando en la implementación el Emulador de teléfonos IP fabricado por esta misma empresa, el cual fue descripto previamente en la sección *Emulador de Teléfono IP*.

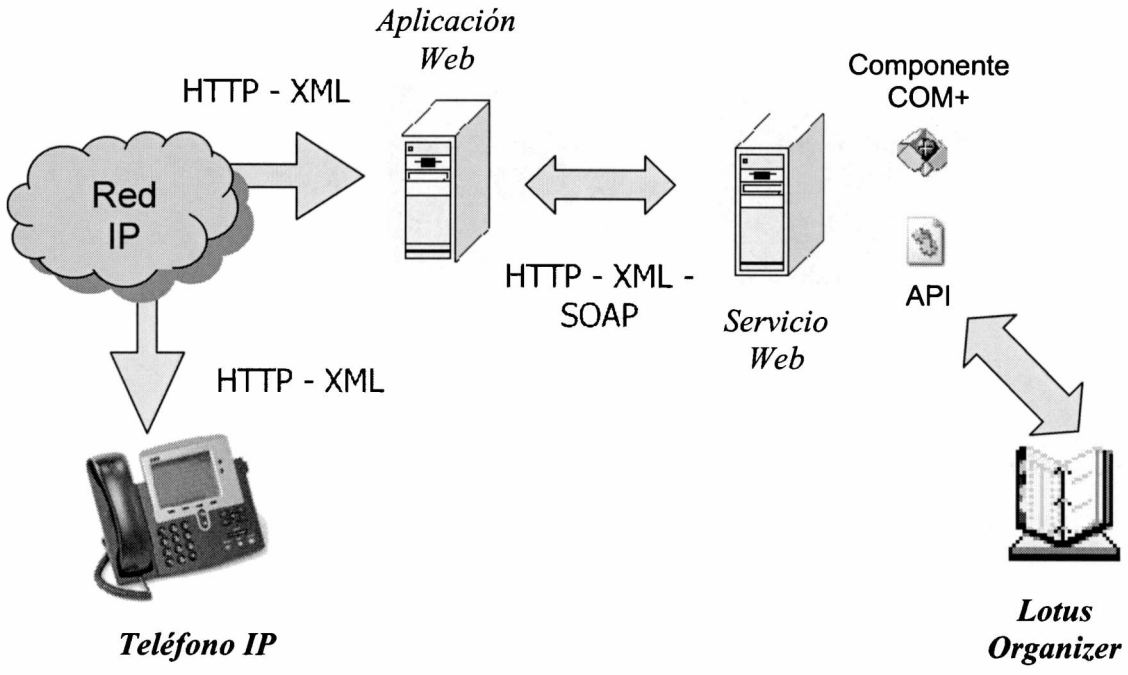


Figura 31. Componentes y relaciones.

Servicio Web

El servicio Web accede a la información almacenada por la herramienta de gestión de tiempos para consultarla y modificarla. Fue implementado en tecnología .NET, usando el lenguaje de programación C#.

La información consultada es procesada y transformada para que luego pueda ser interpretada por la aplicación Web, quien la formateará en XMLs interpretables por el teléfono IP. De similar manera, el servicio Web permite recibir de parte del teléfono IP, un requerimiento de modificación, para lo cual transforma la información de esa solicitud y accede a los datos de la aplicación de gestión de tiempos para modificarlos.

El acceso a los datos lo realiza mediante un componente COM+ desarrollo para tal efecto, el cual expone un conjunto de métodos que realizan las operaciones de consulta y modificación de la información, haciendo uso de la Lotus Organizer API.

Este componente COM+ expone los métodos para realizar ABMC sobre la información de la aplicación de gestión de tiempos. Tiene métodos para manejar sesiones, consultar un listado de Secciones del Lotus Organizer, modificar los datos contenidos en esas secciones tales como, por ej., para la sección Directorio, provee métodos para crear un contacto, consultarlo y modificarlo. Cada uno de estos métodos utiliza las funciones y tipos de datos provistos por la Lotus Organizer API.

Acá se listan la mayoría de los métodos desarrollados en el componente COM+:

InitSession (inicializa una sesión con la API)
TermSession (termina la sesión abierta con un InitSession)
OpenFile (abre el archivo con la información del Lotus Organizer)
CloseFile (cierra un archivo abierto del Organizer con OpenFile)
GetSections (obtiene las secciones del Organizer)
OpenSection (abre una sección)
CloseSection (cierra una sección abierta)
GetSectionClass (obtiene información de las secciones de a una por vez)
SectionRecordCount (retorna la cantidad de registros de una sección abierta)
ModifyRecord (actualiza datos de un registro abierto)
RecordClose (cierra un registro)
RecordDelete (elimina un registro)
RecordFindFirst (ordena los registros de una sección en base a un criterio especificado y devuelve el primero de ellos)
RecordFindNext (devuelve el siguiente registro del ordenamiento creado con un RecordFindFirst)
RecordOpenByUniqID (abre un registro especificado por un id)
GetField (obtiene los datos de un campo especificado)
ContactInsert (crea un nuevo contacto con la información pasada como parámetro)
AppointmentInsert (crea una nueva cita con la información pasada como parámetro)
ErrorCode (retorna un código de error si la API pude determinar la razón de la última falla)

El flujo de una aplicación para manejo de estas funciones es el siguiente [29].

- Inicialización de la API, estableciendo un inicio de sesión que retorna un id de sesión
- Abrir el archivo cotenedor de la información del usuario, utilizando el id de sesión obtenido en el paso anterior.
- Abrir una sección o consultar la lista de secciones disponibles
- Abrir un registro dentro la sección, pudiendo obtener información de los campos incluidos en el registro
- Una vez finalizada la operatoria, se debe cerrar el registro, la sección, el archivo y, finalmente, la sesión.

Así, por ejemplo, para crear un contacto se deben seguir los siguientes pasos:

- Abrir la sesión
- Abrir el archivo
- Abrir la sección Directorio
- Crear un registro válido para la sección Directorio e insertarlo en ella.
- Cerrar sección, archivo y sesión.

Para implementar este Servicio Web se desarrollaron un conjunto de clases que representan los elementos del Lotus Organizer y permiten el flujo de la aplicación

mencionado usando la API. Dentro de este conjunto de clases, las principales se describen a continuación:

- *LotusWS*: Clase que representa la Interfaz de acceso al Lotus Organizer. Sería el Servicio Web propiamente dicho con los siguientes WebMethods:
  - LotusWs (el constructor)
  - GetSections
  - GetRecord(string section, string rid)
  - RecordDelete(string section, string rid)
  - InsertContact(string section, string first, string last, string phone, string company, string email)
  - ModifyContact(string section, string rid, string first, string last, string phone, string company, string email)
  - ModifyAppointment(string section, string rid, int day, int month, int year, int hour, int minute, int duration, string desc)
  - InsertAppointment(string section, int day, int month, int year, int hour, int minute, int duration, string description)

- *LotusOrg*: Clase que representa al Lotus Organizer. Es instanciada en el LotusWS y genera en cada método el flujo de ejecución de la operación sobre los datos, es decir, abrir la sesión con el Lotus Organizer, abrir el archivo de datos, hacer las operaciones y luego cerrar los elementos necesarios.

- *Session*: Clase que representa una sesión con el Lotus Organizer.

- *OrgFile*: Clase que representa un archivo de datos del Lotus Organizer.

- *Section*: Clase que representa una sección del Lotus Organizer (Agenda, Directorio, Notas, etc.).

- *OrgRecord*: Clase que representa un registro genérico del Lotus Organizer, independientemente de a qué sección pertenece. Tiene métodos para manejar estos registros, es decir, búsquedas, borrado, modificación, etc.

- *Record*: Clase abstracta para manejar los registros específicos de las distintas secciones.

- *RecordDirectorio*: Clase que representa un registro de la sección Directorio.

- *RecordAgenda*: Clase que representa un registro de la sección Agenda.

Para el modelado de las clases se tomó como referencia el difundido lenguaje UML (Unified Modeling Language).

UML es un lenguaje para especificar, visualizar, construir y documentar los elementos de un sistema de software [28]. Es utilizado para entender, diseñar, ver, configurar y mantener información sobre dichos sistemas [39].

UML representa una colección de buenas prácticas de ingeniería que ya han sido probadas satisfactoriamente en el modelado de sistemas grandes y complejos. Es una

parte muy importante en el proceso de desarrollo de sistemas orientados a objetos y usa, en su mayoría, notación gráfica para mostrar el diseño de proyectos de software. La utilización de UML ayuda a que los equipos de proyecto se comuniquen, exploren diseños potenciales y validen el diseño arquitectónico del software [28].

Este lenguaje admite la construcción de una serie de diagramas que permite a los desarrolladores y clientes ver un sistema de software desde diferentes perspectivas y grados de abstracción. Uno de los diagramas posibles es el Diagrama de Clases, el cual muestra una colección de elementos declarativos, tales como clases, tipos, y su contenido y relaciones. Mediante este tipo de diagrama se muestran a continuación las clases del Servicio Web.

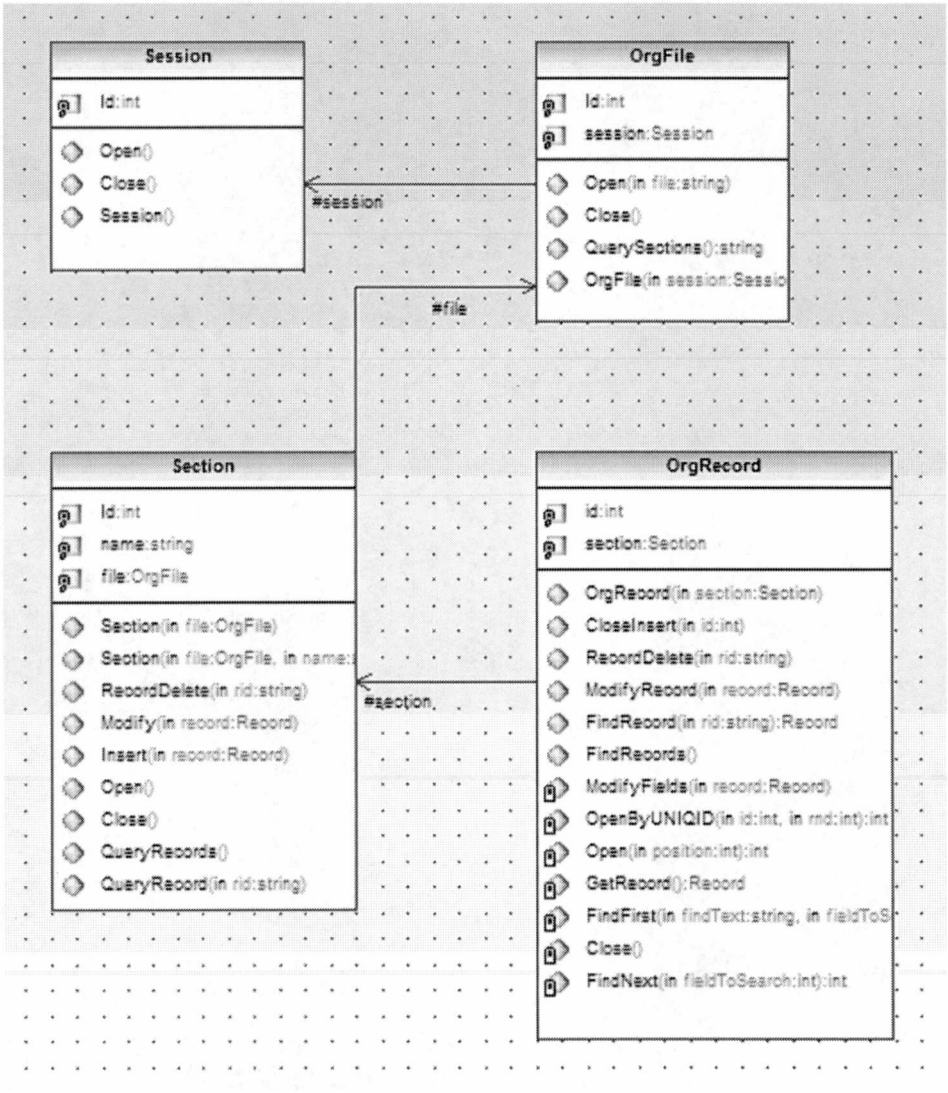


Figura 32. Diagrama de clases del Servicio Web. Principales clases que representan a los elementos del Lotus Organizer.

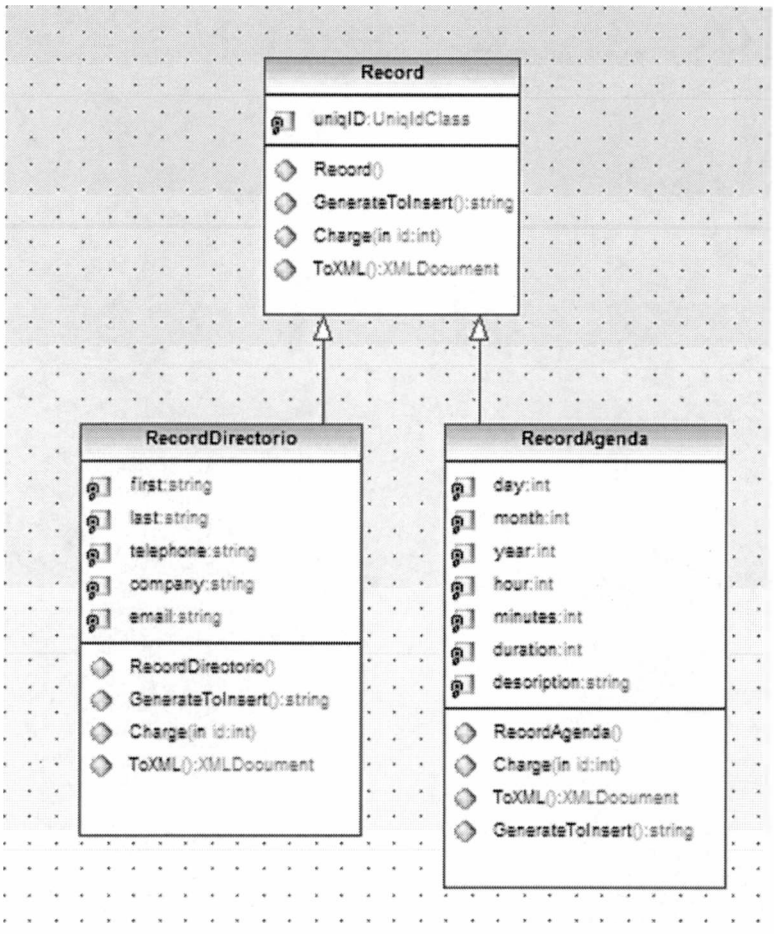


Figura 33. Diagrama de clases del Servicio Web. Clase Record y subclases RecordDirectorio y RecordAgenda.

Aplicación Web

La aplicación Web se encarga de recibir los requerimientos del teléfono IP utilizando el protocolo HTTP. Luego le envía los resultados en forma de XMLs interpretables por el teléfono.

Para lograr estos resultados la aplicación Web se comunica con el Servicio Web y utiliza las funciones provistas por éste, de esta manera, envía y recibe la información almacenada por la herramienta de gestión de tiempos.

La aplicación Web recibe la solicitud del teléfono IP, la interpreta y transforma en una invocación a una función del Servicio Web. Si es una solicitud de consulta de datos, entonces la aplicación Web espera la respuesta del Servicio Web con la información obtenida de la herramienta de gestión de tiempos, la transforma en un XML interpretable por teléfono dándole, además, el formato para mostrarlo en el display del teléfono IP y, finalmente, se la envía a este último.

Así, por ejemplo, cuando desde el teléfono se quiere consultar una cita de la Agenda, el teléfono envía los datos de consulta de la cita ingresados por el usuario, la

aplicación Web toma el requerimiento y lo traduce en una consulta al servicio Web. Éste obtiene la información almacenada en el Lotus Organizer, arma un XML con los datos de la cita obtenidos y se lo envía a la aplicación Web, la cual, finalmente, arma un XML interpretable por el teléfono IP y se lo retorna a este último que muestra en su display la fecha, hora y descripción de la cita.

Para la generación de los XMLs interpretables por el teléfono, se creó una clase padre llamada XMLPhone. Mientras que para el armado de cada uno de los tipos de XMLs, se crearon subclases de XMLPhone, las cuales se listan a continuación:

Objeto: XMLPhone

XmlPhoneDirectory: representando al tag <CiscoIPPhoneDirectory>.

XmlPhoneInput: representando al tag <CiscoIPPhoneInput>.

XmlPhoneMenu: representando al tag <CiscoIPPhoneMenu>.

XmlPhoneText: representando al tag <CiscoIPPhoneText>.

XmlPhoneItem:

XmlPhoneItemDirectory: representando al tag <DirectoryEntry>.

XmlPhoneItemInput: representando al tag <InputItem>.

XmlPhoneItemMenu: representando al tag <MenuItem>.

XmlPhoneItemSoftkey: representando al tag <SoftKeyItem>.

Al igual que el Servicio Web, la Aplicación Web fue implementada en tecnología .NET, usando ASP .NET y el lenguaje de programación C#.

Teléfono IP

En lo que respecta al teléfono IP, el principal punto a tener en cuenta durante el desarrollo de la aplicación, fue la utilización de los XML interpretables por este dispositivo, los cuales, a partir de un requerimiento HTTP, son formateados y enviados mediante la aplicación Web.

Los teléfonos no almacenan ningún tipo de estado cuando cargan una página XML. Sin embargo, pueden cargar el contenido de la página de diferentes maneras, dependiendo de qué tipo de dato (tipo de XML) contiene la página, lo cual determina su comportamiento.

El parser de XML en un teléfono IP de Cisco no es un parseador completo de XML. No se deben incluir tags distintos a los definidos para estos teléfonos. El parseador tampoco funciona con valores de atributos embebidos en los tags, como por ejemplo, no es capaz de parsear el texto `<Title font='arial'>` [9].

Podemos distinguir los siguientes tipos de XML que puede interpretar el teléfono IP [9]:

- CiscoIPPhoneMenu
- CiscoIPPhoneText
- CiscoIPPhoneInput
- CiscoIPPhoneDirectory
- CiscoIPPhoneImage
- CiscoIPPhoneGraphic

- CiscoIPPhoneMenu

Un menú en el teléfono se compone de una lista de ítems de texto, uno por línea. El usuario selecciona un ítem individual de menú de la misma manera que lo hace con los menús propios del teléfono.

La definición de este XML es la siguiente:

```
<CiscoIPPhoneMenu>
      <Title>El título del menú</Title>
      <Prompt>Acción o pregunta a responder con las opciones del
            menú</Prompt>
      <MenuItem>
            <Name>El nombre de cada ítem</Name>
            <URL>La URL asociada con cada ítem del menú</URL>
      </MenuItem>
</CiscoIPPhoneMenu>
```

Este formato de XML permite especificar un título y una pregunta o acción a realizar la cual es utilizada por todo el menú, seguido de una secuencia de objetos MenuItem, cada uno de los cuales tiene un Nombre y una URL asociada. Se permiten hasta un máximo de 32 MenuItems.

El comportamiento de este menú es idéntico a los menús predefinidos, es decir, el usuario navega a través de la lista de ítems y, eventualmente, selecciona uno utilizando las teclas del teléfono. Una vez que el usuario selecciona la opción del menú, el teléfono genera un requerimiento HTTP a la página con la URL asociada en el ítem.

El siguiente ejemplo muestra el menú para la sección Directorio:

```
<CiscoIPPhoneMenu>
      <Title>Menú Directorio</Title>
      <Prompt>Elija una opción</Prompt>
      <MenuItem>
            <Name>Nuevo</Name>
            <URL>http://localhost/lotusweb/directorioNuevo.aspx?s=Directorio
            </URL>
      </MenuItem>
      <MenuItem>
            <Name>Buscar</Name>
            <URL>http://localhost/lotusweb/directorioBuscar.aspx?s=Directorio
            </URL>
      </MenuItem>
</CiscoIPPhoneMenu>
```

- CiscoIPPhoneText

El tipo de XML `CiscoIPPhoneText` es usado para mostrar texto ASCII en la pantalla del teléfono. El texto del mensaje no debe contener ningún caracter de control, excepto saltos de línea y tabs. El teléfono se encarga de controlar la paginación y saltos de línea necesarios.

La definición de este XML es la siguiente:

```
<CiscoIPPhoneText>
      <Title>El título del texto</Title>
      <Text>El texto a ser mostrado cuerpo del mensaje</Text>
      <Prompt>El texto de pregunta o aviso</Prompt>
</CiscoIPPhoneText>
```

Dos campos opcionales aparecen en este mensaje de XML:

- El primero es el campo Title, el cual identifica al texto que es mostrado en la parte superior de la página del display. Si no es especificado, entonces se muestra el campo Name del MenuItem de la última opción de menú seleccionada.
- El segundo es el campo Prompt, el cual identifica al text que es mostrado en la parte inferior de la página del display. Si no es especificado, entonces no se muestra nada en la parte inferior.

- CiscoIPPhoneInput

Cuando un teléfono IP de Cisco recibe un objeto XML del tipo `CiscoIPPhoneInput`, construye una pantalla de input y la muestra. Luego el usuario ingresa los datos en cada ítem de input y, cuando confirma, el teléfono los envía como parámetros en la URL de destino.

La definición de este XML es la siguiente:

```
<CiscoIPPhoneInput>
      <Title>El título de la pantalla</Title>
      <Prompt>Aviso o acción a realizar en la pantalla</Prompt>
      <URL>La URL destino</URL>
      <InputItem>
            <DisplayName>Nombre del campo de input</DisplayName>
            <QueryStringParam>Nombre del parámetro de la
                              URL</QueryStringParam>
            <InputFlags></InputFlags>
            <DefaultValue>Valor por defecto</DefaultValue>
      </InputItem>
</CiscoIPPhoneInput>
```

Los campos `Title` y `Prompt` son opcionales y se comportan como fueron indicados en el tipo de XML anteriormente expuesto.

El requerimiento HTTP se envía a la URL indicada, especificando una lista de parámetros en forma de query string. Estos parámetros son pares Nombre/Valor, uno

por cada ítem de input. Cabe aclarar que los teléfonos IP no usan el método POST de HTTP.

El tag `InputItem` es usado para delimitar cada ítem de la lista. La cantidad de `InputItems` no puede ser mayor a cinco. Cada ítem de input contiene un campo `DisplayName`, el cual es mostrado como pregunta de ítem particular. Además, cada ítem tiene un tag `QueryStringParam`, el cual representa el nombre del parámetro agregado a la URL. También cuentan con un tag `DefaultValue` que permite mostrar un valor por defecto y un tag `InputFlags` que permite restringir el tipo de dato a ingresar, tales como, texto plano, números, campo de contraseña, etc.

El siguiente ejemplo muestra el XML para la generación de los campos de entrada para la búsqueda en la sección Directorio:

```
<CiscoIPPhoneInput>
        <Title>Directorio - Buscar</Title>
        <Prompt>Ingrese datos de búsqueda</Prompt>
        <URL>http://localhost/lotusweb/directorioBuscar.aspx?a=1&s=Directorio
        </URL>
        <InputItem>
                <DisplayName>Nombre</DisplayName>
                <QueryStringParam>first</QueryStringParam>
                <DefaultValue />
                <InputFlags>A</InputFlags>
        </InputItem>
        <InputItem>
                <DisplayName>Apellido</DisplayName>
                <QueryStringParam>last</QueryStringParam>
                <DefaultValue />
                <InputFlags>A</InputFlags>
        </InputItem>
</CiscoIPPhoneInput>
```

- CiscoIPPhoneDirectory

El objeto XML `CiscoIPPhoneDirectory` está incorporado en el teléfono para soportar la funcionalidad de Directorio provista por el teléfono IP de Cisco. Dado que esta funcionalidad existe en el teléfono, está disponible para el desarrollo de aplicaciones para estos dispositivos.

La definición de este XML es la siguiente:

```
<CiscoIPPhoneDirectory>
      <Title>Título del Directorio</Title>
      <Prompt> </Prompt>
      <DirectoryEntry>
            <Name>Nombre de la entrada de Directorio</Name>
            <Telephone>Número de teléfono</Telephone>
      </DirectoryEntry>
</CiscoIPPhoneDirectory>
```

- CiscoIPPhoneImage

El teléfono IP de Cisco tiene un display con un panel de 133 x 64-píxeles el cual está disponible para aplicaciones de servicio como la que se desarrolló durante este proyecto. Cada píxel tiene cuatro posible colores en una escala de grises.

El objeto XML `CiscoIPPhoneImage` permite usar la pantalla del teléfono para mostrar gráficos al usuario.

La definición de este XML es la siguiente:

```
<CiscoIPPhoneImage>
      <Title>Título de la imagen</Title>
      <LocationX>Información de posición del gráfico</LocationX>
      <LocationY>Información de posición del gráfico</LocationY>
      <Width>Información de tamaño del gráfico</Width>
      <Height>Información de tamaño del gráfico</Height>
      <Depth>Número de bits por píxel</Depth>
      <Data>Datos</Data>
      <Prompt>Texto de Prompt</Prompt>
</CiscoIPPhoneImage>
```

Los tags `LocationX` y `LocationY` se utilizan para posicionar el gráfico en el display del teléfono. Mientras que los tags `Width` y `Height` son para especificar el tamaño de la imagen.

Por otra parte, el tag Data delimita un string de dígitos hexadecimales que contienen el valor de los pixels para la pantalla. En los teléfonos IP de Cisco, cada píxel puede tener sólo cuatro valores posibles, por lo cual se pueden empaquetar cuatro pixels en un byte. Cada byte está representado por un par de dígitos hexadecimales.

- CiscoIPPhoneGraphicMenu

Los menús gráficos sirven para el mismo propósito que los menús de texto, es decir, permiten al usuario elegir una URL de una lista. Se suelen usar en las situaciones en que los ítems no pueden ser mostrados en una lista de texto. En estos casos, la información es presentada como un gráfico de mapa de bits.

La definición de este XML es la siguiente:

```
<CiscoIPPhoneGraphicMenu>
      <Title>Título del menú</Title>
      <LocationX>Información de posición del gráfico</LocationX>
      <LocationY>Información de posición del gráfico</LocationY>
      <Width>Información de tamaño del gráfico</Width>
      <Height>Información de tamaño del gráfico</Height>
      <Depth>Número de bits por píxel</Depth>
      <Data>Datos</Data>
      <Prompt>Texto de Prompt</Prompt>
      <MenuItem>
            <Name>Nombre de cada ítem de menú</Name>
            <URL>La URL asociada con cada ítem del menú</URL>
      </MenuItem>
</CiscoIPPhoneGraphicMenu>
```

Los ítems en el menú gráfico tienen un nombre de la misma manera que lo tienen en un menú de texto. A pesar de que ese nombre no es mostrado al usuario, igualmente cumple una función. El nombre del ítem del menú provee el título por defecto usado cuando es cargada la URL del ítem elegido. Si esta página tiene un título propio, entonces éste es usado en lugar del nombre del ítem de menú.

Los tags del XML en el menú gráfico usan la misma definición que `CiscoIPPhoneImage` y `CiscoIPPhoneMenu`. La semántica de los tags es idéntica, aunque se pueden tener hasta doce objetos `MenuItem` dentro de un objeto `CiscoIPPhoneGraphic`.

8.3. Limitaciones y problemas encontrados

La amplia gama de tecnologías utilizadas en este proyecto y su integración desencadenaron en la aparición de algunas limitaciones en las herramientas y, también, en la generación de ciertos problemas.

Uno de los puntos principales en este aspecto, fue conseguir un elemento que sea capaz de reproducir las funcionalidades que posee un teléfono IP, por lo menos en lo que respecta a la comunicación con Internet y aplicaciones Web, ya que no contábamos con un teléfono IP real. De esta manera, se optó por utilizar un emulador provisto de forma gratuita por Cisco Systems. Este emulador gratuito no posee la amplia gama de funcionalidades que tienen otros emuladores no gratuitos, además de presentar algunos errores de funcionamiento.

Por ejemplo, la personalización de los textos de los botones que aparecen en el display del teléfono no está permitida en toda su amplitud, es decir, se pueden dejar las descripciones estándar o generar todas las descripciones personalizadas, pero no se permite hacer una combinación entre ellas. Esto solamente constituyó una limitación, porque sólo afectó al aspecto gráfico que mostraba el display.

Pero surgió como problema grave, un bug que presenta el emulador y que consiste en la no codificación de caracteres especiales durante la generación de las URLs que utiliza para conectarse con una aplicación Web. Así, por ejemplo, cuando se utiliza el emulador para completar datos solicitados y en ellos se ingresan caracteres especiales (por ejemplo, un espacio), la URL resultante que genera el emulador es inválida y, por ende, es rechazada por el servidor Web que la recibe. Ante este problema, se pensó en utilizar otro emulador, aunque no tuvimos éxito, dado que no se encontraron otros gratuitos ni en versión de evaluación, con similares características al utilizado.

Por otra parte, la forma en que el Lotus Organizer almacena la información es a través del manejo de un archivo, por lo cual, para poder implementar una aplicación de este estilo, los archivos de cada usuario deberían ser accesibles por el Servicio Web, por ejemplo, dentro de un recurso compartido, para poder tomar y modificar su contenido. Además, de esta manera, los mecanismos de seguridad de acceso a la información serían más complicados de implementar. Está claro que todo esto mejoraría si la herramienta de gestión de tiempos manejara un esquema de Base de Datos centralizada con toda la información de cada usuario. Por esta razón no se incursionó en la

implementación de un control de acceso estricto de seguridad a la aplicación porque escapaba al conjunto de conceptos que se querían mostrar dentro este desarrollo.

En este contexto, se buscó desarrollar un **prototipo** que sea capaz de **"probar los conceptos"** mostrados en este proyecto y, no sumergirse en la implementación de una aplicación ciento por ciento operativa como un producto final de mercado. Así, a pesar de estas limitaciones y problemas, conseguimos desarrollar un prototipo que vuelque a la implementación los conceptos expuestos.

8.4. Prototipo

Dentro de las funcionalidades que posee la herramienta de gestión de tiempos, se buscaron proveer aquéllas que muestren operacionalmente los conceptos que se trataron en este proyecto y que, además, fuesen útiles para que un usuario las acceda fuera de su red corporativa utilizando el teléfono IP.

Asimismo, se eligieron secciones de la herramienta mediante la cual se pudieran realizar todas las operaciones básicas sobre la información del usuario, es decir, realizar consultas, búsquedas, crear nuevos datos, modificar y eliminar los existentes. De esta manera, quedaron volcadas en el prototipo tanto las consultas como las actualizaciones de la información. Es por ello que una de las secciones elegida fue el Directorio, dado que es una sección que requiere todas estas acciones para su utilización.

Como pantalla inicial de la aplicación, se presenta en el display del teléfono IP la lista de secciones principales que tiene la herramienta de gestión de tiempos, permitiendo elegir una de ellas y así entrar a las funciones que ella brinda. Por ejemplo, en la siguiente figura se ve la lista de secciones y, al elegir la sección Directorio, se obtiene como resultado las opciones que hay dentro de dicha sección.

Figura 34. Visualización de lista de Secciones en el display del teléfono.

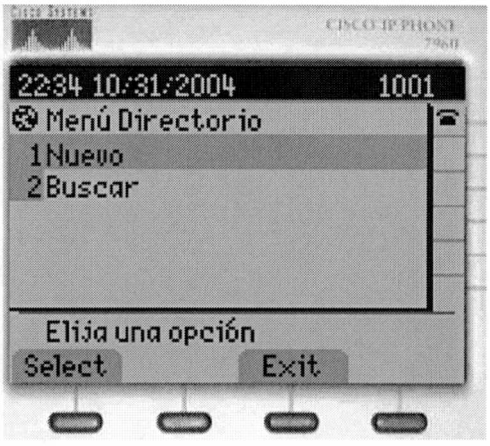


Figura 35. Visualización de la sección Directorio.

Una vez posicionado en la sección elegida, por ejemplo en la sección Directorio, se permite crear un nuevo contacto con sus datos personales, direcciones, etc., como así también se brinda la posibilidad de buscar uno existente por nombre y/o apellido, de manera tal de poder ver sus datos, modificarlo o eliminarlo del Directorio. También se ofrece la opción de efectuar una llamada al número telefónico principal del contacto elegido.

Figura 36. Búsqueda de un contacto en el Directorio y resultados obtenidos.

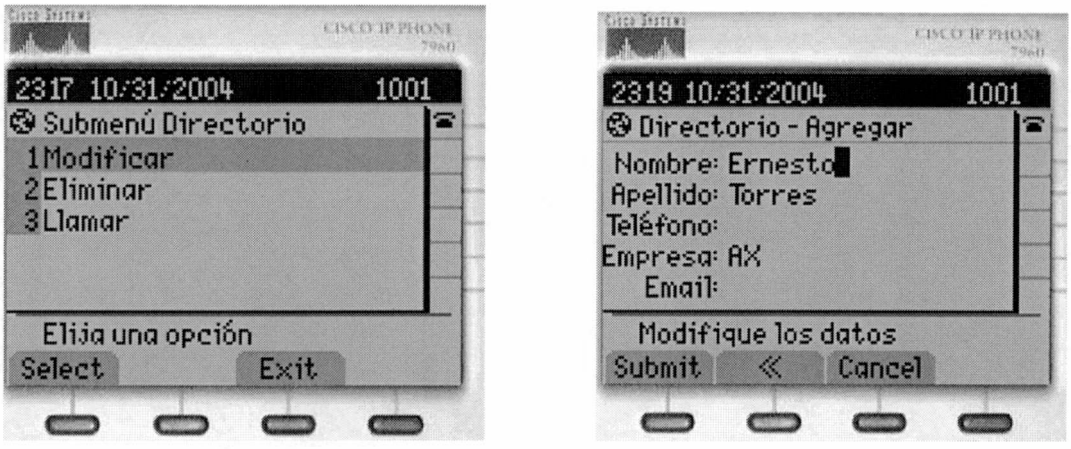


Figura 37. Una vez elegido el contacto se puede ver, modificar, eliminar o hacer un llamado.

Otra de las secciones implementadas en el prototipo es la Agenda. Desde el teléfono IP se pueden crear o consultar citas almacenadas en un rango de fechas.

Cuando se consultan citas de la Agenda mediante búsquedas por fechas, se muestran los resultados coincidentes y se da la posibilidad de elegir una de ellas. Una vez que se selecciona una cita, se permite la opción de ver sus detalles, de modificarla o eliminarla de la Agenda.

Por ejemplo, para crear una cita en la Agenda se muestra la siguiente pantalla en el teléfono IP

Figura 38. Pantalla de creación de una cita de la Agenda

Este conjunto de pantallas del prototipo muestran, en definitiva, cómo se puede manipular la información del Lotus Organizer mediante las páginas enviadas por la aplicación Web usando el display y el teclado del teléfono IP.

9. Conclusión

La utilización de teléfonos IP abre un gran abanico de soluciones posibles para el acceso a información corporativa o global. La creciente preocupación tecnológica para definir estándares y alentar su utilización permiten la expansión de este tipo de soluciones.

Durante el desarrollo de esta Tesis los estándares como XML, SOAP y HTTP tuvieron un papel fundamental, ya que sin ellos no se hubiesen podido integrar las distintas herramientas, fabricantes y plataformas. Así, se lograron integrar los teléfonos IP de Cisco Systems y el Lotus Organizer de IBM, junto con la plataforma .NET de Microsoft, sobre la cual se implementó la solución integradora.

La construcción de una aplicación como prototipo alcanzó para mostrar los conceptos expuestos a lo largo de este trabajo y demostrar que el esquema de integración es viable tecnológica y funcionalmente. De esta manera, se logró acceder a la información manejada por un gestor de tiempos como el Lotus Organizer utilizando un teléfono IP, pudiendo consultar, crear y modifcar los datos almacenados.

La utilización de estándares queda potenciada en la implementación del Servicio Web que hemos desarrollado y que permite hacer extensible este desarrollo para que pueda ser utilizado desde otras herramientas tales como Celulares, PDAs o aplicaciones Web destinadas a otros fines, quizás distintos a los teléfonos IP usados como clientes en este proyecto. Asimismo, queda como extensión a esta Tesis agregar mecanismos de seguridad para el acceso a los datos y el viaje de la información a través de Internet.

De esta manera, existe un amplio espectro de posibilidades para el desarrollo de servicios para teléfonos IP, los cuales pueden acceder a información de variadas fuentes de información gracias al uso de estándares y Servicios Web, dos elementos fundamentales que, junto con la telefonía IP, dictan las tendencias actuales de la tecnología informática.

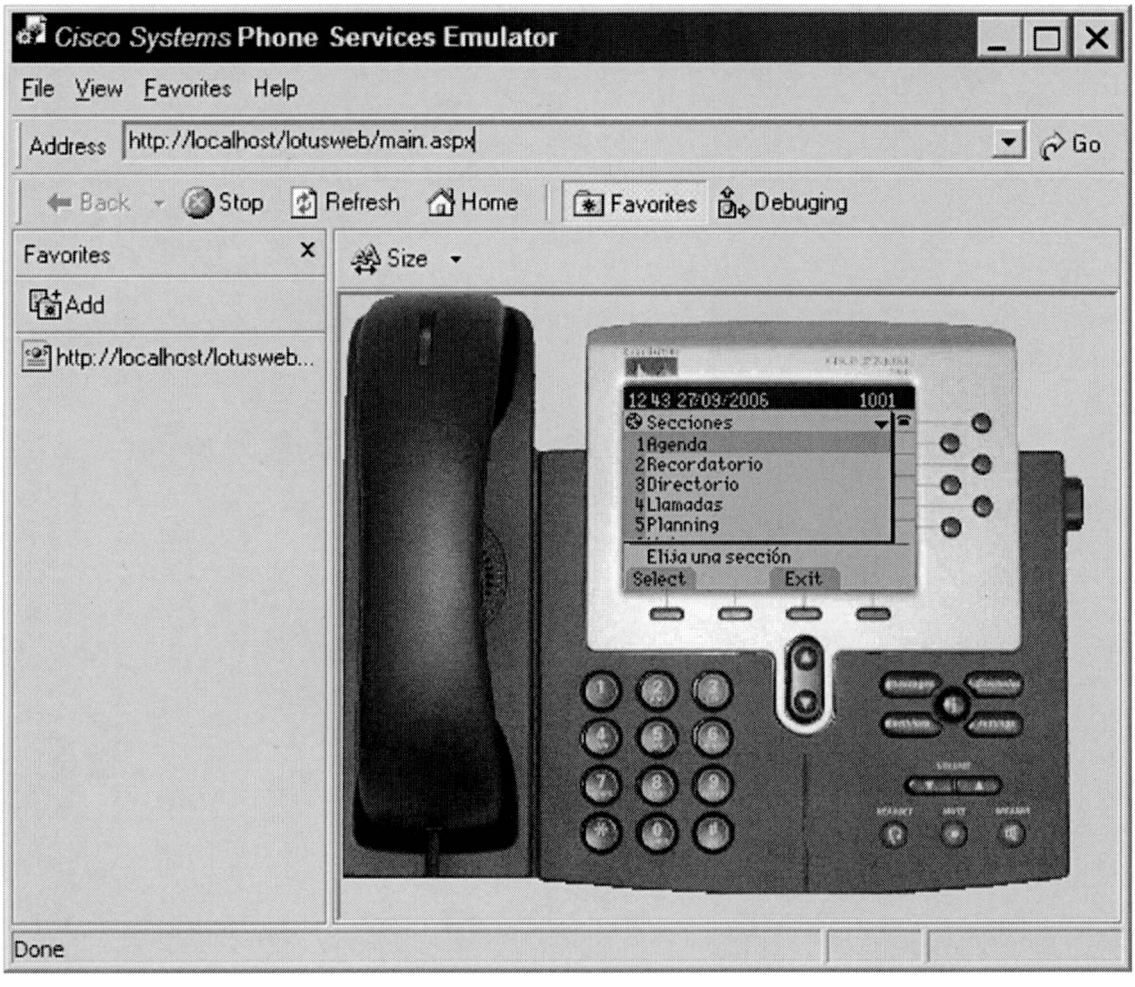
10. Anexo A – Manual del Emulador

10.1. Instalación

La instalación del emulador *CISCO Phone Services Emulator* debe realizarse mediante la ejecución del archivo PSESetup.EXE, el cual instala en el directorio "C:\Archivos de programa\Cisco Systems\Phone Services Emulator" el software del emulador. Este directorio destino puede ser modificado durante la instalación. Además se generará un ícono con la descripción *Phone Services Emulator* en el escritorio de Windows, el cual debe ser ejecutado para ingresar al emulador.

10.2. Uso del Phone Service Emulator

Para empezar a utilizar el emulador se debe hacer doble click en el ícono del escritorio, esto mostrará una pantalla similar a la siguiente.



Dentro del emulador, se puede indicar una url en el campo *Address* lo cual permite establecer una conexión por HTTP y mostrar el resultado en la pantalla del teléfono. Cabe aclarar que la respuesta enviada por la aplicación web consultada debe ser un XML compatible con los usados por el teléfono IP emulado.

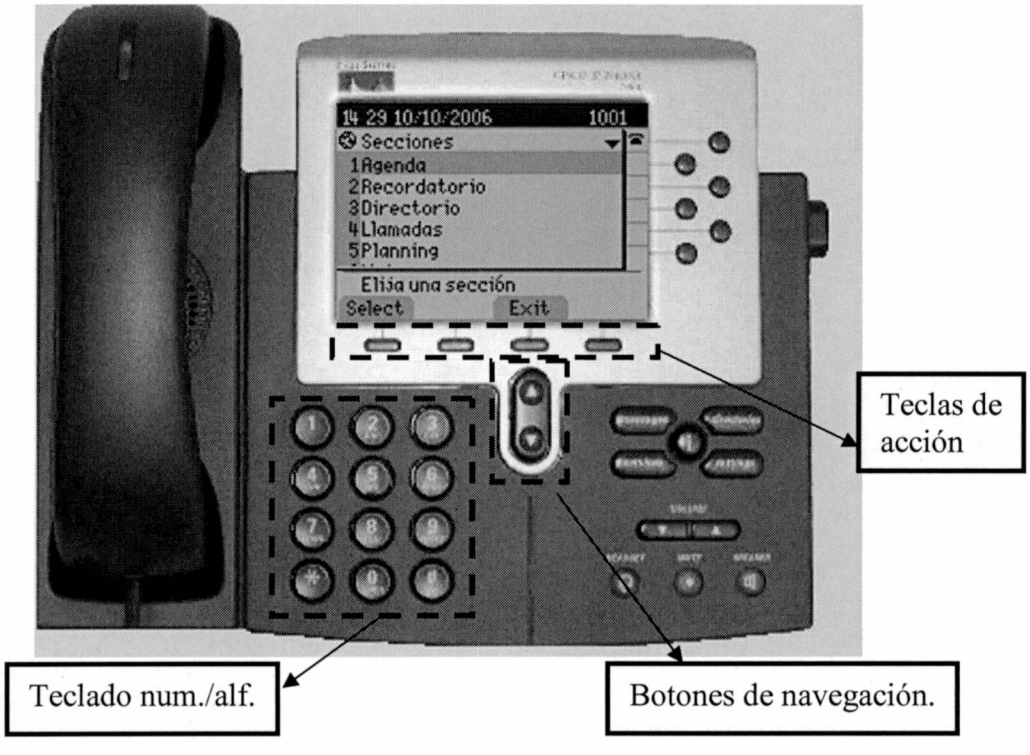
Tal como en un navegador, se pueden agregar direcciones a los favoritos y usar acciones como *"Refresh"*, *"Stop"*, *"Home"* y *"Back"*.

Eligiendo la opción del menú *File* → *Properties*, se puede configurar un limitado conjunto de propiedades tales como la página de inicio del emulador.

Mediante estas funcionalidades se puede navegar por aplicaciones hechas para teléfonos IP utilizando la pantalla y el teclado del teléfono.

10.3. Uso del teclado del teléfono del emulador

Este emulador permite el uso de algunas teclas del teléfono emulado. Para ello se debe posicionar el cursor del mouse sobre la tecla y hacer click, esto generará una acción en la pantalla del teléfono como, por ej., el ingreso de caracteres.



Botones de navegación: permite la navegación entre las opciones que se muestran en la pantalla del teléfono. Por ej., en la pantalla anterior, se pueden bajar y subir entre las Secciones que muestra el menú.

Teclas de acción: permiten elegir las opciones que se muestran en la parte inferior de la pantalla del teléfono. Estas opciones van cambiando de acuerdo a la pantalla que se muestra. En este ejemplo, se podría presionar la primera tecla de la

izquierda ocasionando la selección de la Sección sombreada ("Agenda"), o bien, la tercera tecla para salir.

Teclado numérico/alfanumérico: permite el ingreso de caracteres que son mostrados en la pantalla del teléfono, de manera similar a la que se realiza en un teléfono celular. Además, en el caso de tener un menú en pantalla, tal como se muestra en la figura anterior, se puede elegir una opción. Así, por ej., para elegir la Sección Directorio se puede presionar el botón con el número 3.

Para ingresar caracteres en un campo de ingreso de datos, se debe proceder como se detalla a continuación.

Si se quiere ingresar, por ej., la letra "r" para hacer una búsqueda de los nombres que empiezan con r en la sección Directorio, se debe presionar el botón con el número 7. Luego de esto aparecerán en la pantalla un conjunto de caracteres y, presionando repetidamente (dos veces) el botón con el número 7 hasta llegar a la letra "r", quedará ingresada la misma.

Al presionar el botón con el número 7

Luego presionar el botón con el número 7 tres veces

De la misma manera, se pueden ingresar a continuación otros caracteres.

10.4. Manual de uso de la aplicación.

Para poder ingresar a la aplicación desarrollada a través de Internet hay que poner en el campo *Address* del emulador la siguiente dirección web: http://163.10.20.60/lotusweb/main.aspx.

La primera pantalla que aparece es la siguiente:

En esta pantalla se listan las distintas secciones del Lotus Organizer, a las cuales se puede acceder a través de los botones de navegación o del teclado alfanumérico.

Las únicas secciones desarrolladas a modo de ejemplo fueron las secciones *"Agenda"* y *"Directorio"*

Una vez seleccionada una de estas secciones aparece la siguiente pantalla:

En ambos casos podemos elegir por agregar un nuevo registro a la sección seleccionada o buscar un registro ya ingresado.

Ahora seguiremos con la sección *"Agenda"*, para el caso de la sección *"Directorio"* es similar.

Si elegimos la opción *"Nuevo"* nos aparece la siguiente pantalla:

Una vez ingresados los valores correspondientes a cada uno de los campos, mediante la tecla de acción *"Submit"* se procede a realizar el ingreso en el Lotus Organizer.

Se si elige la opción *"Buscar"* aparece la siguiente pantalla:

Una vez ingresados los parámetros de búsqueda, mediante el botón de acción *"Submit"* se realiza la búsqueda y aparece la siguiente pantalla de resultados:

Aquí tengo la posibilidad de ver los resultados ordenados de forma ascendente o descendente mediante las teclas de acción correspondientes.

Si elegimos uno de los resultados pasamos a la siguiente pantalla:

Aquí tenemos la posibilidad de modificar el registro seleccionado, eliminarlo o simplemente ver su información asociada.

En caso de que elijamos la opción de *"Modificar"* nos aparece la siguiente pantalla:

Aquí podemos realizarle todas las modificaciones necesarias al registro y luego mediante las teclas de acción podemos almacenar las modificaciones.

En caso de elegir la opción *"Ver"* nos aparece la siguiente pantalla:

Si elegimos la opción de *"Eliminar"* nos aparece la siguiente pantalla de confirmación:

11. Bibliografía

[1] **Asp.Net** – Programmer's Reference. Bell, Clark, Elmhorst, Gibbs, Homer, Lee, Milner, Narkiewicz, Rehan, Schenken. Wrox Press Ltd,. 2001.

[2] **ASP.NET Database Programming**. Jason Butler - Tony Caudill. Wiley, John & Sons, Incorporated, Diciembre 2001.

[3] **ASP.NET with C#**. Vijay Mukhi. (http://www.vijaymukhi.com/). The Basics, BPB. Publications, 2001

[4] **Ayuda de Windows 2000/Windows Server 2003** – Microsoft Corporation.

[5] **Best practices for Web services**. Holt Adams, Dan Gisolfi, James Snell, Raghu Varadan. IBM, Octubre 2002.

[6] **Building XML Web Services For The Microsoft .NET Platform**. Scott Short. Microsoft Press, Febrero 2002.

[7] **Cisco CallManager Administration Guide**. Cisco Systems, Inc., 2001.

[8] **Cisco IP Phone Models 7960 and 7940 User Guide**. Cisco Systems, Inc - 2000-2001.

[9] **Cisco IP Phone Services Application Development Notes.** Cisco Systems, Inc., 2004 – 2005.

[10] **Cisco IP Telephony Glossary of Terms**. Cisco Systems, Inc., 2001.

[11] **Cisco IP Telephony Network Design Guide**. Cisco Systems, Inc., 2001.

[12] **Cisco Voice and IP Communications. (www.cisco.com)**. Cisco Systems, Inc.

[13] **COM and .NET Component Services**. Juan Ignacio Velazquez Gonzalez. O'Reilly, 2002.

[14] **COM: Component Object Model Technologies**. MSDN – Microsoft Corporation – 2002.

[15] **COM+ Overview for Microsoft Visual Basic Programmers.** Ted Pattison – Microsoft Developer Network. Microsoft Corporation. Febrero 2000.

[16] **Consumiendo Web services desde .NET**. Víctor García Aprea. Revista Code Nro. 01, Enero 2004.

[17] **Developing Cisco IP Phone Services**. Darrick Deel, Mark Nelson, Anne Smith. - Cisco Press, Febrero 2002.

[18] **El lenguaje de programación C#**. José Antonio Gonzalez Seco. 2001.

[19] **Guía para programadores de .Net Framework**. Microsoft Corporation.

[20] **Introducción a Web Services con herramientas de Desarrollo Microsoft.** Danysoft Internacional. Oscar González Moreno, Abril 2002.

[21] **Introduction to ASP.NET**. .NET Framework Developer's Guide. Microsoft Developer Network. Microsoft Corporation.

[22] **IP Telephony - Today/Tomorrow/Ever?**. Ericcson, July 1998.

[23] **IP Telephony versus PSTN**. Sonus Networks, 2000.

[24] **Learning XML**. Erik T. Ray. O'Reilly Media Inc., Enero 2001.

[25] **Lotus Organizer Help – Release 4.1**. Lotus Development Corporation.

[26] **MicroFocus Cobol** – www.microfocus.com.

[27] **Microsoft Component Services. A Technology Overview**. Microsoft Corporation. 1998.

[28] **OMG-Unified Modeling Language**. Object Management Group, Inc., Septiembre 2001.

[29] **Organizer API Help**. Lotus Development Corporation.

[30] **Pasarse a C#, ¿si o no?**. Ángel "Java" López. Revista .Code Nro.4, Agosto 2004.

[31] **PDC#**. Papper de Robert Schmidt. (http://msdn.microsoft.com/voices/deep.aspx). Microsoft Corporation, Julio de 2000.

[32] **Programming C#**. Jesse Liberty. O'Reilly, Febrero 2002.

[33] **Programming Microsoft .NET**. Jeff Prosise (Wintellect.) Microsoft Press, Mayo 2002.

[34] **Proyecto Mono**. Héctor Facundo Arena. Revista Code Nro.6, Octubre 2004.

[35] **Real World XML Web Services**. Yasser Shohoud. Flyheart.Com, Septiembre 2002

[36] **Standard ECMA-335 Common Language Infrastructure (CLI).** 3rd edition ECMA International, June 2005.

[37] **The Programmable Web: Web Services Provides Building Blocks for the Microsoft .NET Framework**. Article of MSDN Magazine by Mary Kirtland, 2000.

[38] **The State of IP Telephony**. TechRepublic, Inc., Noviembre 2002.

[39] **The Unified Modeling Language Reference Manual**. James Runbaugh, Ivar Jacobson, Grady Booch. Addison Wesley Longman, Inc., Diciembre 1998.

[40] **Turn your office phone network into an application platform with VoIP and XML**. Paper. Veronika Megler, Julio 2002.

[41] **Visual Studio .NET: Build Web Applications Faster and Easier Using Web Services and XML**. Dave Mendlen. MSDN Article, Septiembre 2000.

[42] **Voice over Internet Protocol**. Web Proforum Tutorials. The International Engineering Consortium.

[43] **Web Service Architectures**. Papper de Judith M. Myerson. Web Services Architect, Enero 2002.

[44] **Web Services Developer Center Home**. (http://msdn.microsoft.com /webservices/). Microsoft Developer Network. Microsoft Corporation.

[45] **World Wide Web Consortium (W3C).** http://www.w3.org/TR.